Escuela Superior de Ingeniería

QUIZ LIBRE: Software de entretenimiento basado en un juego de preguntas y respuestas para plataforma Nintendo DS.

Ingeniería técnica en Informática de Gestión

Alumno Agustin Llamas Ruiz

Cádiz, 2012

Escuela Superior de Ingeniería

QUIZ LIBRE: Software de entretenimiento basado en un juego de preguntas y respuestas para plataforma Nintendo DS.

- **Departamento:** Ingeniería informática
- **Autor del proyecto:** Agustin Llamas Ruiz
- **Directores del proyecto:** Antonio García Domínguez y Manuel Palomo Duarte

Índice general

Índice de figuras

Licencia

Capítulo 1

Introducción

En la época que vivimos los videojuegos significan una gran parte del ocio de la población, aportando en ello un gran mercado tanto de variedad como de calidad que, los usuarios eligen según intereses y economía. En este marco donde solo las empresas tienen los recursos, principalmente, para la realización de los juegos, se está alzando un grupo de personas que se dedican a realizar bibliotecas y ayudas para los programadores mas amateur para la realización de juegos caseros para PC y últimamente está recibiendo un gran empuje la programación de videojuegos para consolas, como es este caso, por ejemplo la Nintendo DS.

Este proyecto, aprovechando el empuje que supone actualmente en la red estos juegos, quiere proporcionar a los usuarios para la consola portátil de Nintendo, la Nintendo DS, un pequeño juego de preguntas y respuestas parecido al famoso juego del Trivial, donde cuatro jugadores en la misma consola podrán competir para saber, quien tiene más conocimientos sobre los temas que más les guste, ya que el juego será personalizable por y para los usuarios.

1.1. Objetivos

- **Creación de un juego educativo:** Uno de los objetivos de este proyecto es la creación de un juego basado en un juego de mesa de preguntas y respuesta donde hasta cuatro jugadores puedan medir su intelecto en una gran variedad de temas, además de registrar un ranking donde los jugadores puedan intentar superarse tanto a ellos mismo, como a otros jugadores que hayan jugado anteriormente al juego.
- **Personalización del juego:** El juego cuenta con un sistema de gestión de preguntas y respuestas aparte, donde los jugadores podrán personalizar los temarios y categorías del juego, además de agregar o eliminar los temas que crean oportunos, para dar una mayor variedad al juego.
- **Interacción del juego con una base de datos SQL:** Para la gestión de, tanto las preguntas como los tableros del juego, se ha utilizado una base de datos SQL gestionada por la librería libre **sqlite3**[11], donde a través del código se pueden realizar consultas, eliminación, inserción o actualización de la información de la base de datos en tiempo real durante la ejecución. La base de datos estará aparte del juego para que se puedan modificar los datos externamente.

1.2. Alcance

El proyecto de este juego realizado para la Nintendo DS está pensado para dar a entender, que no solo se pueden programar juegos de forma amateur para PC, sino que también se puede realizar tanto aplicaciones como juegos para otras plataformas como son las consolas, que gracias a comunidades de internet que investigan y realizan aplicaciones para las consolas, se ha conseguido que las consolas no sean tan estáticas como intentan las empresas de videojuegos, sino que también se puedan realizar tanto juegos como aplicaciones personales que puedes incluso distribuir en internet para que otras personas puedan usarlas.

Puesto que las empresas no están muy conformes con la realización de estas aplicaciones, ya que la relacionan con la piratería, las aplicaciones abren un campo nuevo en el área de la programación para consolas que dan libertad al usuario de usar sus periféricos como deseen y necesiten.

1.2.1. Identificación del producto

Este proyecto tiene como finalidad elaborar un juego, utilizando bibliotecas libres de manejo de la consola Nintendo DS, de un juego para la misma llamado QUIZ LIBRE[9], basado en un juego de mesa de preguntas y respuestas parecido al juego Trivial Pursuit® de la compañía Hasbro. Además de un programa de gestión para gestionar las preguntas y respuestas del juego para que los usuarios tengan total libertad en los temarios que desean agregar.

1.2.2. Características generales del juego

Quiz Libre[9] es un juego escrito en lenguaje C++[5] con programación orientada a objetos, donde los jugadores podrán ejecutarlo desde una consola Nintendo DS con un cartucho vendido en tiendas para la ejecución de software casero. Al ser una consola privada de Nintendo se han reunido comunidades de personas en internet para colaborar en la elaboración de bibliotecas para el manejo de la consola. Gracias a estas bibliotecas, más concretamente la biblioteca libre libnds[2] y la biblioteca llamada PAlib, Se ha podido desarrollar este juego.

Las bibliotecas estaban diseñadas para el manejo completo de la consola, como la muestra de imágenes por pantalla, sprites, ejecución de sonido y demás funcionalidades. El juego QUIZ LIBRE[9] usa todas estas funcionalidades para la ejecución del mismo. También usa la biblioteca llamada SQLITE3[11], que es una biblioteca libre para la creación, y gestión de una base de datos en SQL para código C++[5] y contiene toda la información que necesita el juego para funcionar, desde la información de los tableros y sus casillas, como las categorías y preguntas que contiene el juego.

También se incluye en el proyecto un programa de gestión escrito en visual basic .NET donde, añadiendo la biblioteca de manejo de SQLITE3[11], se gestionan todas las preguntas de la base de datos del juego, para que cualquier persona que lo desee, añadir a la base de datos preguntas y categorías para su posterior uso.

1.2.3. Aplicaciones del Software

Este juego está realizado, además del disfrute de las personas de jugarlo y personalizarlo, para aprender a manejar, a través de programación C++[5] orientado a objetos, la consola de Nintendo portátil llamada Nintendo DS, no solo para la consola actual, sino también para las consolas futuras como es la Nintendo 3DS.

También sirve de ejemplo de manejo de una base de datos a través de código C++[5] con sentencias SQL ejecutadas desde el mismo.

1.3. Definiciones, abreviaturas y acrónimos

- **PAlib**: Biblioteca libre para el manejo de la Nintendo DS. Es una biblioteca que trabaja sobre las funciones de libnds[2] que tiene funciones de más bajo nivel.
- **Libnds**[2]: Biblioteca libre para el manejo de la Nintendo DS de más bajo nivel. Pertenece al conjunto de bibliotecas de programación de Devkitpro[1].
- **Stylus**: Lápiz usado en la consola Nintendo DS para jugar con ella.
- **SQLite3**[11]: Biblioteca libre para el manejo de un fichero DB, que es una base de datos en SQL para programas que no tengan soportes para una base de datos propia. Esta base de datos puede ser utilizada en una gran variedad de lenguajes de programación.
- **GBA**: Abreviación de Game Boy Advance, consola antecesora de la NDS fabricada por Nintendo en el año 2000
- **Tile**: Medida de imagen donde un tile corresponde a un cuadrado de la imagen de 8 x 8 pixeles.
- **NDS**: Abreviación de Nintendo DS, consola fabricada por Nintendo.
- **Backgrounds**: Imágenes que se colocan atrás de todas las imágenes de las pantallas para dar un fondo a los programas.
- **Sprites:** Cada una de las imágenes de una pantalla, que representan un objeto del programa.
- **Clase singleton**: Clase creada de la cual solo se crea un objeto de la misma que se comparte por todo el programa. Para ello se crea un puntero que siempre apunta al objeto creado en primer lugar.

1.4. Visión general de la memoria

Este documento sigue, de una manera más o menos fiel, las pautas recogidas por varios profesores del Departamento de Lenguajes y Sistemas Informáticos en el documento Recomendaciones para la realización de la Documentación del Proyecto de Fin de Carrera. A continuación se describen de forma general los capítulos que componen la memoria:

- **Introducción**: Identificación de objetivos, alcance y aplicaciones del proyecto. Vistazo general de la memoria.

- **Planificación temporal**: Desarrollo del calendario que se ha seguido a la hora de realizar el proyecto.
- **Juegos de mesa y juegos de preguntas y respuestas:** Introducción a los juegos de mesa y como son los juegos basados en preguntas y respuestas.
- **Videoconsolas y Nintendo DS**: Breve introducción a las videoconsolas, descripción de las características generales de cada generación de los videojuegos desde su primera aparición, ademas de las propiedades de la consola fabricada por Nintendo llamada Nintendo DS.
- **Desarrollo para la consola Nintendo DS y la librería PAlib**[6]: Descripción de las bibliotecas que está basado el juego para poder ejecutarse en la consola.
- **Fase de análisis del juego**: Idea, especificación de requisitos, modelado y diagramas que componen el juego.
- **Fase de diseño del juego**: Tras realizar el análisis del juego se procede al diseño de los elementos que compondrán el juego como son la interfaz y las clases que componen la aplicación.
- **Implementación**: Parte de más contenido de la memoria donde se explican todo el desarrollo de la aplicación basándose en los dos capítulos anteriores.
- **Conclusiones**: Valoración global del trabajo realizado en el proyecto, posibles mejoras y ampliaciones
- **Pruebas**: Descripción de las distintas pruebas que se han realizado para comprobar y validar los componentes del juego.
- **Bibliografía y referencias**: En este capítulo se indican todas las fuentes de información consultadas para realizar el proyecto.
- **Apéndices**: Se incluyen como apéndices un listado de software utilizado en la elaboración del proyecto, el manual programación en PAlib[6] y el texto de la licencia bajo el que se libera este documento, que es la GFDLv1.3.

Capítulo 2

Planificación

En este apartado se retrata la organización llevada a cabo por el proyecto durante la realización del mismo. Se puede diferenciar cuatro apartados principales en la organización del proyecto y fue realizado entre finales del año 2010 hasta mediados del año 2012.

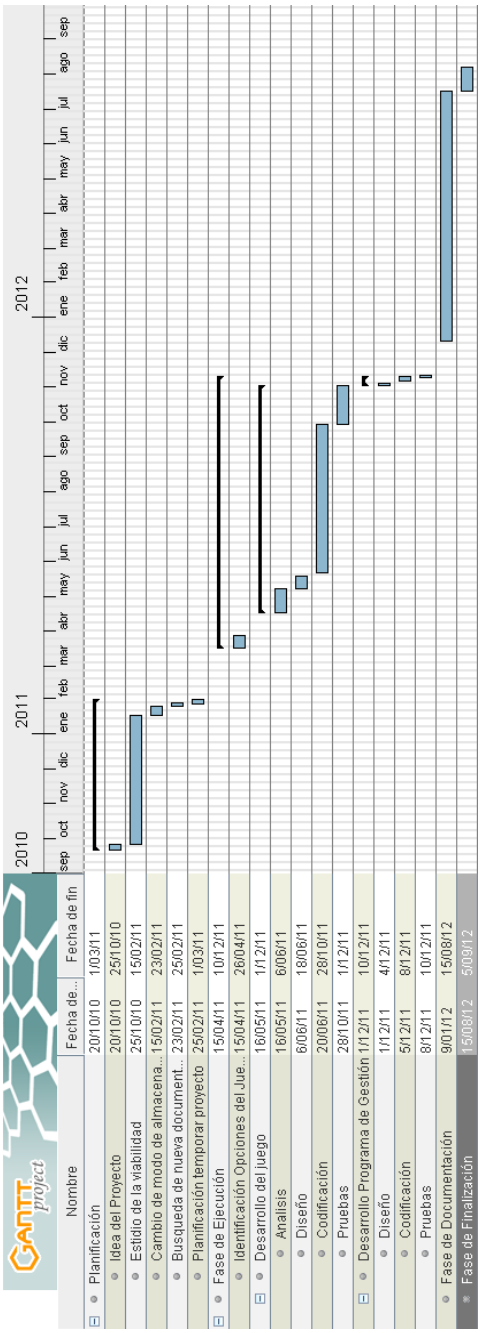


Figura 2.1: Diagrama de gantt

- **Planificación:** Tiempo dedicado a pensar que proyecto realizar y como. Se consultó varias fuentes e ideas para el proyecto, llegando a la conclusión en hacer un videojuego de la Nintendo DS, ya que era un tema actual que la gente trataba en internet.

    - **Idea del Proyecto:** Con la idea de realizar un juego de la Nintendo DS y que sea tanto educativo como divertido para que los jugadores, se decide realizar la adaptación de un

juego de mesa muy conocido de preguntas y respuestas, ya que no existía aun en la consola citada ningún juego que tuviera algún parecido al trivial.

- **Estudio de la viabilidad:** Después de un estudio exhaustivo de las posibles alternativas a utilizar para la realización del proyecto, se decide usar una biblioteca desarrollada por aficionados franceses llamada PAlib[6] que, junto a la biblioteca libnds[2] desarrollada en el pack de devkitpro[1], facilita mucho la labor de manejar la consola con funciones simples. Al comienzo se intentó una programación estructurada del juego, ya que no había mucha información sobre otro método de programación con la Nintendo DS, pero podo después de un estudio más profundo, se pudo concretar que es posible realizar el proyecto con programación orientada a objetos para mayor simplicidad y mejor funcionamiento. Sobre el almacenamiento de las preguntas, se comenzó a utilizar XML con la biblioteca tinyXML.
- **Cambio de modo de almacenamiento:** Por complejidad del lenguaje XML para el almacenamiento masivo de información, se decide la utilización de otro método de almacenamiento que fuera más práctico. Se selecciona la opción de utilizar la biblioteca libre llamada sqlite3[11] que, gracias a unas funciones simples y concisas y un fichero a modo de base de datos, se puede crear tablas y vistas para el almacenamiento, tanto de los tableros como de las categorías que pertenecen al juego para después utilizarlo durante el mismo.
- **Búsqueda nueva documentación:** Con el propósito de incluir la base de datos en el proyecto, se dispuso a buscar nueva documentación para poder usar la biblioteca sqlite3[11] en el proyecto y la mejor forma de usar dicha biblioteca.
- **Planificación temporal del proyecto:** Una vez ya recolectados todos los recursos a utilizar para la realización del proyecto, se realiza una planificación temporal para la realizar el proyecto, así como las partes que iba a constar el mismo.

- **Fase de ejecución:** Proceso de realización del proyecto desde el análisis del mismo hasta las pruebas realizadas al proyecto. Se ha realizado dos programas para el proyecto: El juego en lenguaje C++[5] para la Nintendo DS y un programa de gestión escrito en Visual Basic .NET para la gestión de las preguntas del juego.

  - **Identificación opciones del juego:** Estudio e identificación de las posibles acciones principales del juego, cuales el jugador podrá seleccionar al inicio del juego, para después realizar la programación correspondiente a cada opción.
  - **Desarrollo del juego:** Realización del análisis, programación y pruebas del juego con las bibliotecas antes mencionadas.
    - **Análisis:** Identificación de las clases del proyecto con sus propiedades y métodos, distribución de tareas a las clases y sus relaciones. Estudio de las diferentes opciones del juego así como su configuración.
    - **Diseño:** Realización del diagrama de clases y de interacción del proyecto, así como el modelo Entidad-Relación de la base de datos. Diseño de los diferentes sprites e imágenes necesarias para el juego, también el diseño de las diferentes pantallas y su distribución para la elaboración del menú del juego, como las imágenes que saldrán durante el proceso del juego. Búsqueda del sonido del juego, Así como el diseño de las categorías con sus preguntas que se utilizarán en el juego.
    - **Codificación:** Realización de la programación del juego de todas las clases y métodos del mismo, realización de la base de datos en la consola de sqlite3[11].

    - **Pruebas:** Testeo del juego tanto a nivel de clases como a nivel general. El testeo del juego fue llevado a cabo por varias personas jugando al mismo.
  - **Desarrollo Programa de Gestión:** Realización del análisis, programación y pruebas del programa de gestión de las preguntas del juego con el programa de Visual Studio.
    - **Diseño:** Diseño de los formularios cuales constara el programa de gestión, así como las funcionalidades de los mismos.
    - **Codificación:** Realización de la programación de los botones y funcionalidades de los formularios. Conexión con la base de datos perteneciente al juego para una modificación directa.
    - **Pruebas:** Pruebas básicas de las funcionalidades de los formularios para su correcto funcionamiento, tanto a nivel de formularios como de base de datos.

- **Fase de Documentación:** Realización de la documentación que explica todo el proceso llevado las especificaciones detalladas del proyecto.
- **Fase de Finalización:** Fase de finalización y entrega del proyecto, depuración de errores en la documentación y culminación del trabajo.

Capítulo 3

Juegos de mesa y videojuegos

Un juego de mesa es un juego que requiere una mesa para jugarse o un soporte similar y que es jugado generalmente por un grupo de personas alrededor de él. Aunque el azar puede ser una parte muy importante en este tipo de juegos, también los hay en los que son necesarios estrategia y razonamiento para poder jugar.

Por su naturaleza, en general los juegos de mesa no conllevan actividad física, aunque existen algunos que implican levantarse de la mesa y realizar actividades fuera de ésta ya sea por castigo o recompensa, en este caso estos serían juegos de mesa pero no limitados a la misma.

Con el empuje de los videojuegos, cada vez adaptan más juegos de mesa a las consolas y PC de cualquier índole, esto ocurre por varias razones:

- Gracias a los videojuegos se pueden organizar las partidas más rápidamente y sin necesidad de montar el tablero y que ocupe espacio.
- Se tiene una mayor organización a la hora de realizar las partidas, además se evitan equivocaciones en el juego.
- Con la llegada de internet, los videojuegos de juegos de mesa se han adaptado para poder jugar con gente en la red, ya sea de tu mismo país o de otro.

3.1. Juegos de preguntas y respuestas

Uno de los juegos más populares entre los juegos de mesa, son los juegos de preguntas y respuestas. La finalidad del juego es competir entre los jugadores para saber quién tiene más conocimientos sobre temas, ya sean generales o específicos, y determinar un ganador en la partida, que será quien más preguntas ha respondido correctamente.

Uno de los juegos más famosos de la actualidad es el Trivial Pursuit© realizado en 1979 por Scott Abbott, y desde entonces, ha sido el juego de referencia para muchos otros juegos de mesa basado en preguntas y respuestas. Quiz Libre es una simulación del juego pero en plataforma Nintendo DS.

Capítulo 4

Videoconsolas

Una videoconsola es un pequeño sistema electrónico que está diseñado para ejecutar juegos desarrollados en un computador personal o servidor. Al igual que los ordenadores personales, pueden adoptar diferentes formas y tamaños; de este modo, pueden ser de sobremesa, es decir, requieren ser conectadas a un televisor para la visualización del videojuego, y a la red eléctrica para su alimentación o bien el dispositivo electrónico videoconsola portátil, que cuenta con una pantalla de visualización integrada y una fuente de alimentación propia (baterías o pilas).

4.1. Generaciones de las consolas

En la Industria de los videojuegos, las videoconsolas han sido clasificadas en distintas generaciones. Esta clasificación la determina su tiempo de lanzamiento y la tecnología existente en ese momento, o por otro lado, algunas generaciones están señaladas por un número determinado de bits, los cuales determinan el ancho de bus del procesador. A continuación se describen las generaciones que existen actualmente y sus características comunes:

Primera Generación

- Transcurre desde 1972 hasta 1977.
- En 1972 se lanzó la primera videoconsola de sobremesa, llamada Magnavox Odyssey.
- Utilizaban pantallas vectoriales.
- Se creó el juego arcade Pong.

Segunda Generación

- Inicia a finales de los años 70 hasta la primera mitad de los años 80.
- El dominio absoluto fue de Atari.
- Se incluye por primera vez en la historia una CPU de 16 bits.
- Empresas japonesas como Sega o Nintendo hacían incursiones a nivel doméstico.



Tercera Generación

- Primeras consolas portátiles.
- Crisis de los videojuegos en 1983 y las consolas pasan a monopolio japonés.
- Las consolas tenían 8 bits.




Cuarta Generación

- Inicia en 1987.
- Salen consolas de 16 bits de CPU.
- Esta generación destaca por los chips gráficos añadidos al cartucho, como el Super FX y SVP y las ampliaciones de hardware de Mega Drive: Mega CD y Mega 32X.
- Aparecen conceptos como multitarea, multimedia, gráficos vectoriales, etc...




Quinta Generación

- Gran variedad de fabricantes que presentaron diversos equipos con características semejantes a la de un ordenador personal.
- Aparecen las consolas de 32 bits y 64 bits.
- Se trata de una generación que supuso el paso de los 2D a los entornos tridimensionales 3D.




Sexta Generación

- Llamada "la era de los 128 bits".
- Comienza a finales del siglo XX.
- Todas las consolas de sobremesa de sexta generación poseen mandos ergonómicos, memorias externas, y, la diferencia más importante, conexiones de internet y red para jugar en línea o en una conexión cerrada.

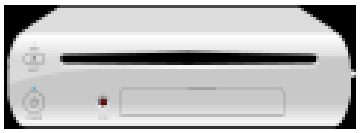


Séptima Generación

- Inicia a finales del 2005 hasta la actualidad.
- Aparecen juegos renderizados de forma nativa con resoluciones de alta definición.
- Reproducción de contenido multimedia HD
- Integración de controladores con sensores de movimiento, así como palancas de mando.

Octava Generación

- La octava generación comienza con el lanzamiento de Nintendo 3DS, el 25 de febrero de 2011.
- Nintendo ha anunciado la sucesora de su consola de sobremesa, la Wii U que se estrenara en 2012/2013.



4.2. Consolas Portatiles

A diferencia de las consolas normales, las consolas portátiles tienen la peculiaridad de que tanto los controles, la pantalla, los altavoces y la alimentación están integrados en un aparato portátil que el usuario puede llevar a donde desee para jugar.

Se puede diferenciar tres tipos de consolas portátiles:

- **Videojuego electrónico portátil**: No tienen cartuchos intercambiables, discos ópticos, discos magnéticos, tarjetas de memoria, etc... O no son reprogramables y de un solo juego normalmente.
- **Videoconsola portátil:** Permite diferentes videojuegos a través de cartuchos intercambiables, discos ópticos, discos magnéticos, tarjetas de memoria, etc.
- **Videoconsola portátil de código abierto**: Permiten ser programadas libremente. Parten de la misma idea de una consola portátil, pero permiten ser totalmente reprogramadas, y tienen una práctica ausencia de juegos oficiales. Funcionan a través de programas gratuitos creados en comunidades de internet.

El primer videojuego portátil que aparece en el mercado con su propia pantalla LCD es un minijuego de Mattel llamado Mattel Auto Race en 1976 y desde entonces las consolas portátiles han estado cogiendo fama y seguidores, no solo por su posibilidad de poderse llevar a cualquier parte, sino por sus funcionalidades cada vez mayor que se le puede dar a los aparatos.

4.2.1. Nintendo DS

La videoconsola Nintendo DS es una consola portátil desarrollada por la compañía japonesa Nintendo para multimedia y videojuegos lanzada al mercado en el año 2004. Las letras DS significan Dual Screen, que es la característica principal de la consola, al igual que una de las pantallas sea táctil. Hasta la fecha se han desarrollado varias versiones de la consola agregando nuevas funcionalidades o potencia a la misma, así como mejorando la calidad de los juegos y posibilidades que la consola brinda a sus usuarios. La consola para abreviar fue llamada NDS.

Actualmente existen en el mercado una amplia gama de consolas DS que vamos a enumerar a continuación:

- **Nintendo DS**: Fue la primera consola sacada al mercado con las características antes mencionadas.

Figura 4.1: Nintendo DS. Fuente: wikipedia.org

- **Nintendo DS Lite**: Se empezó a fabricar en el año 2006 para suceder a la NDS anterior. Las únicas modificaciones que se aplicaron a la consola fueron de estética y la posibilidad de elegir entre cuatro intensidades de brillo de las pantallas.

Figura 4.2: Nintendo DS Lite. Fuente: wikipedia.org

- **Nintendo DSi**: Se empezó a comercializar en el año 2008 después del éxito de la NDS Lite. Se aplicó una serie de mejoras como un indicador de estados de la consola a través de LEDs, se le añade dos cámaras interactivas para utilizarlas en los juegos o realizar fotografías, una interfaz mejorada al estilo de la consola superior Nintendo Wii, navegador de internet, conexión a la tienda de Nintendo y la posibilidad de subir fotos a internet.

Figura 4.3: Nintendo DSi. Fuente: wikipedia.org

- **Nintendo DSi XL**: El mismo año que la NDSi, Nintendo creó una nueva versión de la consola igual a la anterior con la salvedad de que eran más grandes las pantallas de la consola, todo era para que los clientes pudieran elegir, cual consola les interesaba más.

Figura 4.4: Nintendo DSi XL. Fuente: wikipedia.org

- **Nintendo 3DS**: En el año 2011, Nintendo saco la última versión de la NDS como consola portátil de la compañía. Esta nueva consola traía consigo una novedad importante, pantallas 3D sin necesidad de gafas para ver las tres dimensiones. Puede reproducir fotos y videos en 3D, un nuevo sistema de intercambio de información llamado StreetPass y algunas prestaciones más que las antecesores no daban.

Figura 4.5: Nintendo 3DS. Fuente: wikipedia.org

Hardware de la consola

Teniendo en cuenta la gran variedad de versiones que tiene la consola Nintendo DS, para crear aplicaciones para ella necesitaremos tener en cuenta que las nuevas tienen más capacidad que las antiguas, así que cuando creemos la aplicación debemos decidir, si la aplicación se podrá ejecutar en

cualquier versión o, en cambio, que solo se ejecute en las nuevas. En este caso se da algunas características hardware de la videoconsola NDS, que pueden resultar necesarias conocer para desarrollar aplicaciones para cualquier versión, ya que son las características de la consola más veterana:

- La resolución de las pantallas LCD es de 256 x 192 píxeles cada una, también se podrían medir en unidades gráficas usadas en los videojuegos, en este caso la medida de las pantallas serían de 32 x 24 Tiles.
- Dispone de dos procesadores ARM: un ARM9 principal, y un co-procesador ARM7; de 67MHz y 33 MHz, respectivamente, con 4 MB de memoria principal. El ARM7 se encarga de manejar la salida de audio y la pantalla táctil mientras que el procesador principal maneja los gráficos y el resto de procesamiento incluyendo el cómputo de 3D.
- El sistema hardware 3D permite transformaciones y luces, transformaciones de coordenadas de textura, mapeado de texturas, transparencias alfa, anti-aliasing, cel shading y z-buffering. El sistema es capaz teóricamente de manejar 120000 triángulos por segundo, a 60 frames por segundo (FPS). Además del límite de 2048 triángulos por frame a 60 FPS. El sistema está diseñado para renderizar sobre una pantalla la escena 3D, aunque existen algunas posibilidades hacerlo sobre ambas pantallas, esto provocaría una pérdida de rendimiento significativa.
- Permite comunicaciones inalámbricas Wi-Fi con un punto de acceso estándar, o bien con otra consola Nintendo DS. Al conectarse a Internet, se puede acceder a la red Nintendo Wi-Fi para poder competir con otros usuarios.
- En las consolas más antiguas existen dos ranuras para introducir los juegos: la más pequeña está situada arriba y se utiliza para poder jugar a los videojuegos de la propia consola Nintendo DS, la más grande que se encuentra abajo sirve para introducir juegos de la videoconsola portátil antecesora de Nintendo, Game Boy Advance, permitiendo retro compatibilidad y seguir disfrutando de juegos antiguos en la nueva videoconsola, en las más nuevas solo tiene la ranura superior, ya que los juegos de la GBA se ejecutan desde el software.

Limitaciones de la consola

A la hora de programar para la consola NDS, debemos tener en cuenta también ciertas limitaciones que tiene la consola a la hora de ejecutar aplicaciones, puesto que, al ser una consola portátil que tiene ya tiempo a sus espaldas, no tiene mucha potencia y por ello debemos limitarnos a la hora de ejecutar o mostrar simultáneamente objetos en pantalla o cargarlas en memoria. Estas limitaciones son:

- **Imágenes en 256 colores**: Aunque se puede poner imágenes en 8 bits y 16 bits, estas imágenes consumen mucha memoria RAM tenerlas en pantalla y solo podemos tener una simultáneamente por pantalla, así que se recomienda mejor usar imágenes en doscientos cincuenta y seis colores para tener una mayor variedad a la hora de darle vistosidad al juego.
- **Cuatro background por pantalla**: Esto quiere decir que solo podemos tener simultáneamente cuatro fondos por pantalla de la consola. Al igual que se recomienda poner imágenes en doscientos cincuenta y seis colores colores los background no deben de ser de mucha calidad para poder jugar con variedad de fondos.

- **128 sprites por pantalla**: Al igual que la limitación de cuatro fondos por pantalla, tampoco podemos poner más de ciento veintiocho sprites por pantalla. Estos sprites se pueden modificar y transformar con un elemento llamado rotsets que se le asignan a los sprites para realizar transformaciones como, zoom, rotaciones y transparencia. Solo se tiene treinta y dos rotsets disponibles por pantalla así que hay que administrarlos entre los sprites que están. Los sprites deben tener una resolución determinada para que la imagen no se distorsione al cargarla, donde el mínimo de un sprite seria 8 x 8 y el máximo 64 x 64. La tabla de los posibles tamaños seria:

	8	16	32	64
8	8x8	16x8	32x8	
16	8x16	16x16	32x16	
32	8x32	16x32	32x32	64x32
64			32x64	64x64

- **Ocho canales de sonido**: La consola solo dispone de ocho canales para la reproducción de sonido simultaneo, están reservados principalmente para sonido Mod pero se puede cambiar según las exigencias que tenga el juego.

Hardware adicional para cargar aplicaciones Para la ejecución del juego en la consola portátil de Nintendo solo se puede realizar de dos maneras, o bien te instalas uno de los diferentes emuladores existentes en internet, que se instalan en el ordenador para la emulación de juegos y software de la consola, o bien probar el juego directamente en la consola.

Para poder ejecutar un juego o aplicación que se desarrollan por aficionados en la consola directamente, se necesitan un hardware determinado, que emula los cartuchos oficiales, con una memoria de almacenamiento que se pueda gestionar directamente para poder meter o borrar aplicaciones. Este hardware consiste en un cartucho idéntico a un cartucho oficial con una ranura para una tarjeta microSD, donde colocamos dentro un firmware determinado de la marca del cartucho para la carga de homebrew. En el misma tarjeta de almacenamiento se meten las aplicaciones directamente que queremos ejecutar y gracias a la emulación del firmware del cartucho podemos ejecutarlos como si fuera software oficial.

En el caso de este juego hay un pequeño problema, puesto que el juego la información la adquiere de una base de datos externa al mismo, no se puede utilizar emuladores por no poder especificar directamente la ubicación de la base de datos, por ello solo podemos ejecutarlo desde la consola directamente.

Figura 4.6: Cartucho que ejecuta programas caseros en la consola directamente almacenados en una microSD que se le introduce directamente. Fuente: Palib.info

Capítulo 5

Desarrollo en Nintendo DS

Actualmente hay una gran demanda en el área de las consolas y videojuegos y cada vez se quiere sacar más partido a los productos que adquirimos, pero no siempre los fabricantes y compañías dueñas de las consolas otorgan las funcionalidades que queremos, por ello comunidades en internet se han dedicado al estudio y creación de diferentes librerías para la programación de aplicaciones personales que las compañías no han desarrollado en ese momento. Estas comunidades se han ido incrementando y aglutinando cada día más hasta crear ayudas y aplicaciones que las personas van necesitando y compartiéndolas gratis por internet.

Respecto a la consola Nintendo DS, se han desarrollado varias alternativas, con sus guías y ejemplos, para que los programadores, tanto experimentados como amateur, puedan realizar aplicaciones para compartirlas en la comunidad o simplemente para uso propio. Las librerías que usa este proyecto son libnds[2] y PAlib[6].

5.1. Libnds

Libnds[2], o también conocida en su versión anterior como ndslib, fue creada y mantenida por Michael Noland (de apodo Joat), y Jason Rogers (Dovoto), además de haber contribuido mucha gente en su mejora. Esta librería comenzó a ser la primera alternativa al SDK oficial de Nintendo para DS. Permite crear aplicaciones que pueden ser cargadas por los elementos hardwares propios de la videoconsola Nintendo DS, explicados anteriormente. Con esta librería es posible programar todas las características de la videoconsola como son: pantalla táctil, micrófono, hardware 3D, hardware 2D y protocolo wifi. Esta librería contiene información sobre los registros, abstrayendo la información de las direcciones de memoria.

Esto supone las mismas ventajas e inconvenientes que cualquier otra herramienta de desarrollo que tienen bajo nivel de abstracción. El programador tiene mucha responsabilidad al trabajar con los registros lo que supone tener un gran conocimiento de la plataforma, pudiendo ser esto una ventaja porque otorga mucha flexibilidad y poder exprimir al máximo la funcionalidad de la videoconsola.

La librería libnds[2] se divide en varios archivos que abarcan las diferentes funciones y registros para interactuar con la consola. La parte que maneja la visualización 2D tiene por ejemplo funciones

específicas para trabajar con la VRAM (memoria de video) donde se situarán las paletas, texturas, sprites... también incluye un apartado 3D que ofrece una versión reducida de la famosa librería openGL. Para la programación gráfica es necesario indicar el modo de video antes de poder representar nada, cada modo tiene características diferentes como por ejemplo el número y tipo de fondos, permitir rotaciones de los gráficos, otro servirá para representación en 3D, etc.

Con esta librería se generan dos códigos principales: arm9.cpp y arm7.cpp. Cada uno de los cuales es manejado por el procesador correspondiente, realizando las tareas específicas de cada uno teniendo en cuenta los usos anteriormente descritos. Otra parte importante para la programación haciendo uso de esta librería son las interrupciones. Están vinculadas a los eventos del teclado para la comunicación con el usuario, o con el pintado de buffer sobre la pantalla para manejar temporizaciones o para controlar los frames por segundo de la aplicación, por ejemplo. Cuando sucede la interrupción se guarda en un registro de peticiones de interrupción siendo tarea del programador atenderlas o no.

Con esta aproximación se muestra el nivel de abstracción que usa esta librería y dependerá de cada proyecto si puede venir mejor o peor usarla. De este modo la siguiente librería que se va a explicar se convierte en la alternativa perfecta ya que está basada en libnds[2], pero presenta multitud de funciones de alto nivel para simplificar la programación sin tener que estar trabajando con registros, ni memorias de vídeo ni interrupciones.

5.2. PAlib

PAlib[6], se trata de una librería open source basada en libnds[2]. Está compuesta por muchas funciones que facilitan en gran medida el desarrollo de aplicaciones para la videoconsola Nintendo DS. Al contrario que ocurría con la librería libnds[2] que hacía falta conocer los registros que tiene, con esta librería no se trata con el hardware directamente ya que dispone de funciones de más alto nivel de abstracción, que realizan funciones más complejas que a su vez se comunican con la librería libnds[2]. Esto ha hecho que mucha gente que comienza a hacer aplicaciones para esta videoconsola opte por la librería PAlib[6]. El apodo del creador de esta librería es Mollusk y mantenía en la página web gran cantidad de información pero en los últimos meses al parecer han abandonado el proyecto dejando solo pequeños fragmentos en internet de cómo se maneja.

Posterior a la memoria se añadirá una guía de instalación y manejo con ejemplos de la librería que se ha recopilado antes de que desapareciera toda información sobre ella. Se explicara todas las posibilidades y funciones con ejemplos claros y estrategias para una programación fácil y estructurada.

Capítulo 6

Fase de Análisis del Juego

Este capítulo se centra en la fase de análisis, fundamental en cualquier elaboración que contenga una arquitectura. Se parte de una necesidad y a partir de esta se toman las decisiones necesarias para llevar a la práctica un proyecto real. Como se trata de un desarrollo en el que el autor de la idea es la misma persona que va a llevarlo hasta últimos términos, se va a comenzar desde la base primigenia, es decir, desde la toma de decisiones de qué idea se quiere convertir en juego.

En primer lugar se va a plantear qué tipo de juego se va a realizar y por qué, a continuación se decide el guion del juego, a partir de este comienza la fase de análisis infiriéndose en esta cómo se va a estructurar (opciones del juego, elementos que lo componen,...).

Otra parte importante del análisis, después de modelar el concepto en los distintos objetos que se puede descomponer, consiste en analizar los estados en los que se pueden encontrar estos y cómo transitar de unos a otros. Para ello se usarán técnicas de modelado como UML para la elaboración de diagramas de casos de uso.

Una vez esté analizada y definida la idea se pasará al siguiente capítulo, fase de diseño, que abarcará cada elemento por separado de la estructura y lo acercará más todavía a la realidad del desarrollador aunque sin llegar todavía a la implementación.

6.1. Idea

Para definir lo que se quiere conseguir, se parte de la ventaja de conocer los límites que se poseen. Podemos aprovechar esta condición para definir un tipo de juego acorde con la situación.

Se conocen muchos tipos diferentes de juegos, prácticamente puede haber tantas categorías como ocurrencias distintas se puedan tener. En este caso se ha optado por el desarrollo de un juego basado en un juego de mesa de preguntas y respuestas, con la idea de desarrollar un juego para una de las consolas más compradas de la actualidad y utilizar como referencia uno de los juegos más antiguos y divertidos que reta a los jugadores en pruebas de inteligencia y conocimiento general.

Tanto la representación de los elementos del juego como la interfaz del usuario se realizaran en dos

dimensiones. Se usara la pantalla táctil para el movimiento de las fichas y la selección de respuestas durante el juego y para la selección de opciones en el menú de la interfaz. Este tipo de juego está bastante limitado en diversidad de juego por ello se va a dejar libertad del usuario de personalizar el juego a su gusto para conseguir nuevos retos, tanto personales como para amigos.

6.2. Modelado conceptual

Ya está definida la idea del juego. Esta no abarca todos los detalles, por tanto en esta fase también se asumen decisiones.

6.2.1. Opciones del juego

Este juego se compone de dos opciones, que se tendrá que elegir inicialmente para comenzar desde la interfaz del juego. La primera opción es el comienzo de una nueva partida, donde los jugadores comenzaran de cero en las puntuaciones. En esta opción se requerirá la elección tanto de temario como de tablero a jugar, así como cuántos jugadores y sus nombres participaran en la partida. La segunda opción es la carga de una partida ya existente, donde se cargara la partida guardada anteriormente en la base de datos del juego para su continuación, desde el momento en el que le dio a guardar la última vez.

Otras dos opciones, también incluidas en el menú principal del juego, son el ranking del juego, que es una clasificación de los diez ganadores con más puntuación que se ha jugado en el juego y la última opción son los créditos que simplemente es la muestra de agradecimientos e información del juego.

6.2.2. Fases del juego

El juego se compondrá de un bucle que se repetirán una serie de fases para el transcurso del juego. Inicialmente se realizara la tirada del dado (Automáticamente) una vez que el jugador que sea su turno toque la pantalla inferior, cuando se sabe la tirada del dado se procede a la siguiente fase que es la selección por parte del usuario de la casilla a la que desea mover su ficha, que según la casilla a la que se haya movido puede haber varias alternativas:

- La ficha cae en el centro del tablero o en una casilla blanca, donde automáticamente se volverá a tirar el dado para que el jugador vuelva a elegir una casilla donde desee moverse.
- La ficha cae en una de las casillas principales (Esquinas), donde si acierta conseguirá una porción de la ficha, que tendrá que conseguir un total de seis porciones para ganar la partida.
- La ficha cae en una casilla normal.

En la siguiente fase, tanto si la ficha cae en una casilla normal o en una casilla principal del tablero, se realizará una pregunta de la categoría según el color de la casilla. Cada color está asociado con una categoría de la base de datos, así que se seleccionara automáticamente una pregunta de la dicha categoría con sus respuestas y el jugador deberá responderla.

Si el jugador responde correctamente la pregunta propuesta, conservara su turno y se volverá a tirar el dado, sino se pasara al siguiente jugador para que realice su jugada. Si la casilla que se ha respondido la pregunta es una casilla principal del tablero y se respondió correctamente, se proporcionara al jugador una porción del queso del color correspondiente a la casilla que ha respondido la pregunta.

Durante la partida un jugador en su turno puede pausar el juego y guardar la partida para o reanudarla según les convenga o también mirar las categorías asignadas a los colores de las casillas.

El juego finalizará una vez que, cualquiera de los jugadores, consiga las seis porciones de las seis categorías diferentes que componen el tablero.

Una vez finalizada la partida se muestra en pantalla al ganador y lo registra si su puntuación supera a las ya existentes en el ranking y se vuelve al menú principal.

6.2.3. Elementos

En este apartado expondremos los elementos que componen el juego. El elemento principal del juego son los jugadores que es la representación de la persona que está jugando en el juego.

Durante el juego se pueden diferenciar una serie de elementos característicos:

- **El tablero**: Representación del tablero del juego de mesa con sus casillas.
- **Los dados**: Lo dados que se usan para mover la ficha por las casillas.
- **Las fichas**: Pertenecen a cada jugador y las mueven a donde quieran por el tablero, según las tiradas del dado y representa tanto el progreso del jugador en la partida como la situación de éste en el tablero.
- **Las preguntas**: Cada categoría tiene una serie de preguntas asignadas que se eligen aleatoriamente cada vez que un jugador cae en una casilla de color que no sea blanca.

6.2.4. Descripción de los casos de uso

Conociendo como se va a desarrollar la partida, ya se pueden definir los diagramas de casos de uso. A continuación vamos a definir los casos de uso correspondientes al juego:

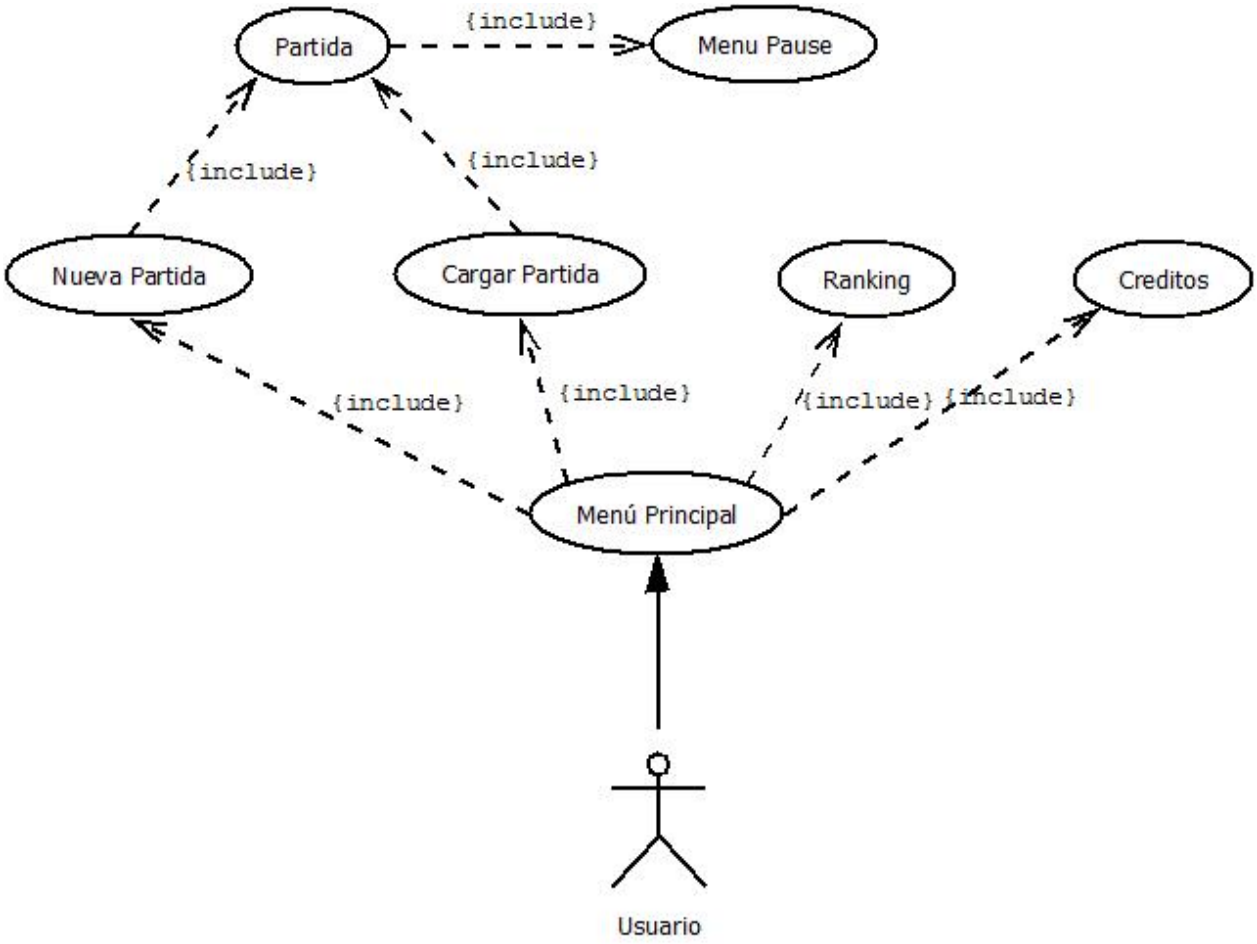


Figura 6.1: Diagrama de los casos de uso

Casos de uso: Menú del juego

El siguiente diagrama es el correspondiente a las diferentes opciones que el jugador puede elegir una vez ejecuta el juego. Son las opciones principales que determina lo que el jugador va a realizar.

El jugador puede realizar cuatro opciones disponibles en el menú para dirigirse a cada una de las partes del juego:

- **Nueva partida**: El jugador desea comenzar una nueva partida en el juego indicando todos los parámetros para el comienzo de la partida. Se iniciara una partida de cero según lo que seleccione el jugador.
- **Cargar partida**: El usuario desea cargar una partida guardada anteriormente en el transcurso de una partida. Los parámetros de la partida son cargados de la base de datos para su continuación. Si no hay ninguna partida guardada se vuelve al menú principal.

- **Ranking**: El jugador desea mostrar el listado de las diez puntuaciones más altas del juego. El sistema cargara el listado de la base de datos y lo muestra en pantalla.
- **Créditos**: El jugador desea ver los créditos del juego. Los créditos es un texto que expone quién ha creado el juego.

En las opciones de carga de partida, ranking y créditos, el jugador no hace nada más que esperar a que el sistema genere por pantalla o la información que desea mostrar o la ejecución de la partida. La ejecución de la partida se explica en otro caso de uso que es común con la opción de nueva partida una vez que se ha introducido los datos de la partida. Pero en la opción de nueva partida el jugador selecciona una serie de parámetros para el inicio de la partida que definimos a continuación.

Caso de uso: Menú principal

- **Descripción:** El sistema muestra al usuario el menú principal del juego, para que seleccione la opción que desea realizar, ya sea comenzar una partida, mostrar el ranking o los créditos del juego.
- **Actores:** Usuario
- **Precondiciones:** Ninguna
- **Postcondiciones:** Ninguna

Escenario principal

1. El sistema muestra en pantalla el menú principal del juego
2. El usuario selecciona la opción "Nueva Partida"
3. El sistema da paso al proceso de ejecución de una nueva partida.

Escenarios alternativos

- **2a.** El usuario selecciona la opción "Cargar Partida", iniciando el proceso de carga de una partida.
- **2b.** El usuario selecciona la opción "Ranking", iniciando el apartado del ranking del juego.
- **2c.** El usuario selecciona la opción "Créditos", iniciando el apartado de los créditos del juego.

Caso de uso: Nueva partida

- **Descripción:** El usuario pulsa el botón de "Nueva Partida" y da comienzo a una partida del juego nueva, seleccionando las configuraciones pertinentes.
- **Actores:** Usuario
- **Precondiciones:** Ninguna
- **Postcondiciones:** Configuración e inicio de una nueva partida.

Escenario principal

1. El usuario desea jugar una partida nueva.
2. El usuario selecciona la opción de nueva partida del menú.
3. El sistema muestra por pantalla la selección de número de jugadores a participar en la partida.
4. El usuario selecciona el número de jugadores pinchando en uno de los cuatro botones de la pantalla con el número correspondiente.
5. El sistema muestra por pantalla la selección de tablero disponibles en el juego. La lista de tableros es cargada de la base de datos.
6. El usuario elige uno de los tableros disponibles.
7. El sistema muestra por pantalla la selección de temarios disponibles en el juego. La lista de categorías es cargada de la base de datos.
8. El usuario elige uno de los temarios disponibles.
9. El sistema muestra por pantalla el teclado para que el usuario introduzca el nombre de un jugador.
10. El usuario escribe el nombre del jugador y le da a continuar.
11. *include* caso de uso: PARTIDA

Repetir pasos 8 y 9 por cada jugador que va a jugar la partida seleccionado en el menú anterior.

Escenarios Alternativos

- ***.a.** El usuario pincha en el botón *atrás*.
    - 1. El sistema vuelve al menú anterior al que está actualmente, si el menú es el de selección de número de jugadores se vuelve al menú principal.
- **9a**. El usuario no escribe ningún nombre.
    - 1. El sistema asigna un nombre por defecto al jugador.

Caso de uso: Cargar partida

- **Descripción:** El usuario pulsa el botón de "Cargar Partida" y da comienzo a una partida del juego ya existente en la base de datos, seleccionando las configuraciones pertinentes.
- **Actores:** Usuario.
- **Precondiciones:** Ninguna.
- **Postcondiciones:** Inicio de una partida ya existente en la base de datos.

Escenario principal

1. El usuario desea continuar una partida guardada con anterioridad.
2. El usuario selecciona la opción del menú principal "Cargar Partida".
3. *include* caso de uso: PARTIDA.

Escenarios alternativos

- **3a.** El sistema no encuentra ninguna partida guardada.
    - El sistema vuelve al menú principal.

Caso de uso: Ranking

- **Descripción**: El usuario pulsa el botón de *Ranking* donde se muestra una lista de jugadores con puntuaciones formando el ranking del juego.
- **Actores**: Usuario
- **Precondiciones**: Ninguna
- **Postcondiciones**: Muestra por pantalla la lista del ranking del juego.

Escenario principal

1. El usuario desea observar el ranking del juego.
2. El usuario selecciona la opción del menú "Ranking".
3. El sistema muestra por pantalla el ranking del juego.

Caso de uso: Créditos

- **Descripción**: El usuario pulsa el botón de *Créditos* donde se muestra un texto con la información del creador del juego.
- **Actores**: Usuario.
- **Precondiciones**: Ninguna.
- **Postcondiciones**: Muestra por pantalla los créditos del juego.

Escenario principal

1. El usuario desea observar los créditos del juego.
2. El usuario selecciona la opción del menú "Créditos".
3. El sistema muestra por pantalla los créditos del juego.

Casos de uso del juego

En los siguientes casos de uso se describen los dos casos de uso que intervienen en una partida:

- **Partida**: Es el caso de uso principal que describe el desarrollo de una partida interactuando con los jugadores.
- **Menú pause**: Muestra un menú de opciones durante la partida.

Caso de uso: Partida

- **Descripción**: El usuario pulsó el botón de *Nueva Partida* o *Cargar Partida* con la intención de jugar una partida donde la configuración depende de si la partida ha sido cargada desde la base de datos, o es una partida nueva.
- **Actores**: Usuario
- **Precondiciones**: Ninguna
- **Postcondiciones**: Se jugará una partida al Quiz Libre.

Escenario principal

1. Usuario toca la pantalla para iniciar su turno.
2. El sistema genera automáticamente la tirada de los dados, suma la cantidad de los mismos y calcula donde puede ir el jugador con la tirada desde su posición.
3. El usuario toca la casilla que desea situarse.
4. El sistema comprueba la categoría de la casilla y selecciona una pregunta aleatoriamente de la base de datos y la muestra en pantalla.
5. El usuario selecciona una respuesta de las tres posibles.
6. El sistema comprueba que es correcta la pregunta, registra la puntuación acorde al acierto.
7. El sistema comprueba que se ha completado las porciones.
8. El sistema muestra por pantalla el ganador y lo registra en el ranking.

Repetir pasos desde el 1 al 6 mientras que un jugador no consiga todas las porciones.

Escenarios alternativos

- **3a**. El usuario pulsa la tecla "Start" de la consola.
  - 1. *include* caso de uso: MENÚ PAUSE.
- **3b**. El usuario pulsa la tecla "Select"de la consola.
  - 1. El sistema muestra por pantalla los nombres de las categorías y su relación con los colores de las casillas, durante este tiempo el juego se mantiene en pause.
  - 2. El jugador pulsa de nuevo la tecla "Select"
  - 3. El sistema reanuda la partida.
- **4a**. La casilla que selecciona es una casilla blanca o el centro del tablero.
  - 1. El sistema vuelve a generar una tirada de los dados y calcula de nuevo las nuevas posibilidades que puede seleccionar el jugador desde la nueva posición.
- **5a**. El usuario no selecciona ninguna respuesta durante un minuto.
  - 1. El sistema da como incorrecta la respuesta.
- **6a**. La respuesta proporcionada por el jugador es incorrecta.
  - 1. El sistema registra la puntuación acorde al error y da permiso al siguiente jugador a comenzar su turno.
- **6b**. La casilla era una casilla principal del tablero y el jugador responde correctamente.
  - 1. El sistema anuncia por pantalla que se ha conseguido una porción del color de la casilla que se realizo la pregunta, registra la nueva puntuación y da permiso al jugador a continuar con su siguiente turno.

Caso de uso: Menú pause

- **Descripción**: El usuario pulsó el botón de la consola *Start* para mostrar el menú de opciones del juego. En este menú se pueden realizar las opciones de "guardar partida", o salir de la misma.
- **Actores**: Usuario
- **Precondiciones**: Solo se puede ejecutar durante la selección de casilla de una partida.
- **Postcondiciones**: Muestra el menú de opciones del juego.

Escenario principal

1. Un usuario desea mostrar el menú de opciones.
2. El usuario pulsa el botón *Start* de la consola.
3. El sistema muestra por pantalla el menú de pausa del juego.
4. El usuario selecciona la opción de *Continuar Partida*.
5. El sistema reanuda la partida.

Escenarios alternativos

- 4a. El jugador pulsa de nuevo la tecla "Start".
  - 1. El sistema reanuda la partida.
- 4b. El jugador selecciona la opción guardar la partida.
  - 1. El sistema da como concluida la partida, registra la partida y todos sus datos en la base de datos y vuelve al menú principal.
- 4c. El jugador selecciona la opción salir de la partida.
  - 1. El sistema da como concluida la partida y vuelve al menú principal del juego.

6.2.5. Modelo conceptual de datos

A continuación se van a especificar los requisitos del sistema y las relaciones entre ellos. Para ello se define un diagrama de clases conceptuales, donde se representa las clases, las asociaciones entre las clases, los atributos que los componen y las relaciones de integridad de las mismas.

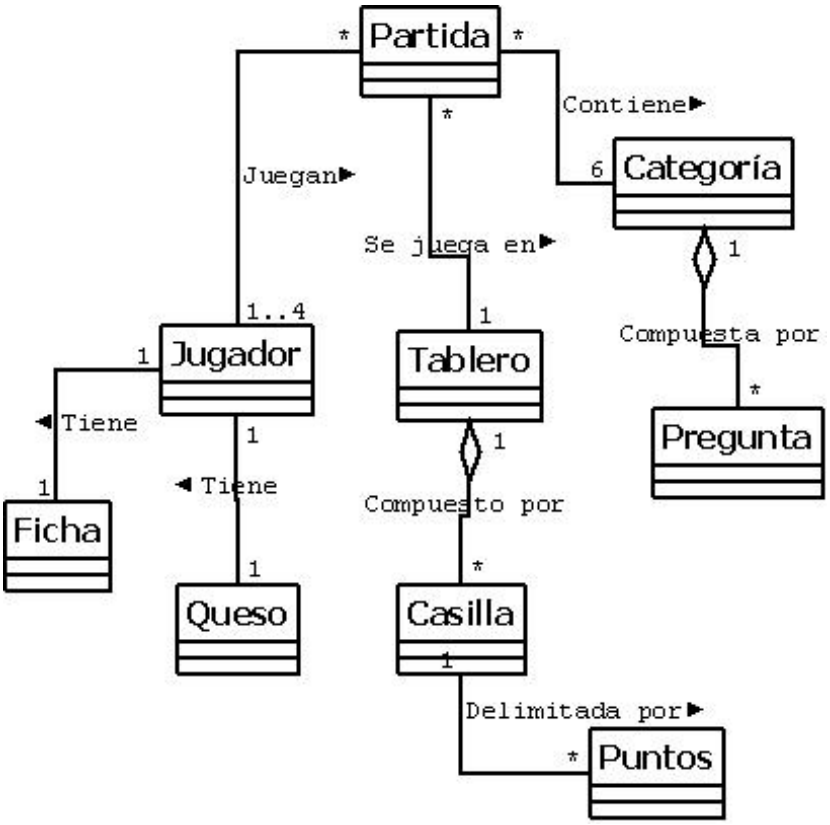


Figura 6.2: Diagrama de clases conceptuales

Descripción de las clases del diagrama

- **Partida**: Clase principal del juego encargada de la gestión completa del transcurso de una partida.
  - **Jugador**: Cada uno de los jugadores que intervienen en una partida.
    - **Ficha**: Cada una de las fichas del juego que representan a cada jugador en el tablero.
    - **Queso**: Ficha incrementada del jugador que indican el marcador del mismo.
- **Tablero**: Clase que representa el tablero y sus opciones del juego
  - **Casilla**: Cada una de las casillas que componen un tablero, donde la ficha puede moverse.
    - **Punto**: Las casillas están delimitadas espacialmente por puntos. Estos indican cual es el espacio de la casilla respecto al tablero.
- **Categoría**: Son los temas diferentes que componen un temario.
  - **Pregunta**: La clase pregunta representa cada una de las preguntas que componen una categoría que a su vez pertenece a un temario.

6.2.6. Modelo de comportamiento del sistema

En este apartado se definen como debe actuar el sistema respecto a la interacción del actor. Consta de dos partes:

- Diagrama de secuencia: Muestra la secuencia de eventos entre actores y el sistema.
- Contratos de operaciones: describen el efecto que produce las operaciones en el sistema

Modelo de comportamiento Nueva Partida

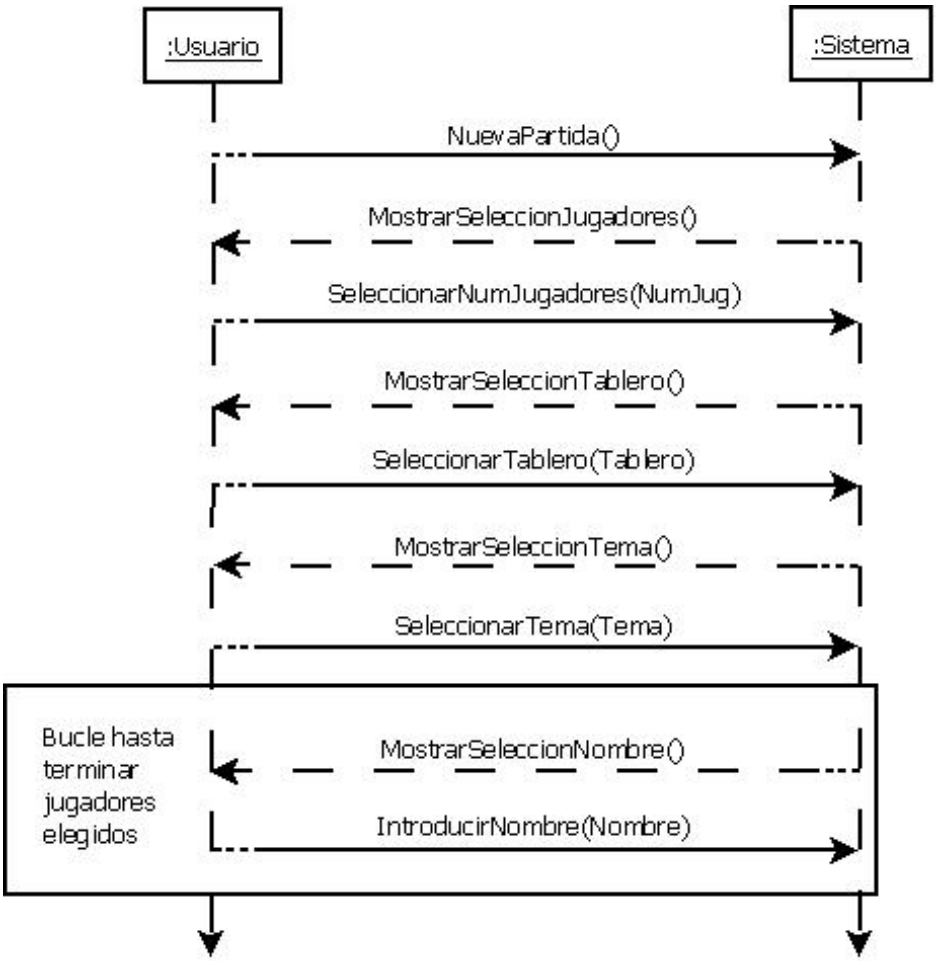


Figura 6.3: Diagrama de secuencia del caso de uso: Nueva Partida

Contratos de operaciones

Operación: NuevaPartida()

Actores: Usuario, sistema.

Responsabilidades: Caso de uso "Nueva Partida".

Precondiciones: Ninguna.

Postcondiciones: Comienzo configuración de una partida.

Operación: MostrarSeleccionJugadores()

Actores: Usuario, sistema.

Responsabilidades: Caso de uso "Nueva Partida".

Precondiciones: Ninguna.

Postcondiciones: Mostrar selección de número de jugadores que jugaran la partida.

Operación: SeleccionarNumJugadores(NumJug)

Actores: Usuario, sistema.

Responsabilidades: Caso de uso "Nueva Partida".

Precondiciones: Ninguna.

Postcondiciones: Selección por parte del usuario del número de jugadores.

Operación: MostrarSeleccionTablero()

Actores: Usuario, sistema.

Responsabilidades: Caso de uso "Nueva Partida".

Precondiciones: Ninguna.

Postcondiciones: Mostrar selección de tablero para la partida.

Operación: SeleccionarTablero(Tablero)

Actores: Usuario, sistema.

Responsabilidades: Caso de uso "Nueva Partida".

Precondiciones: Ninguna.

Postcondiciones: Selección por parte del usuario de un tablero para la partida.

Operación: MostrarSeleccionTema()

Actores: Usuario, sistema.

Responsabilidades: Caso de uso "Nueva Partida".

Precondiciones: Ninguna.

Postcondiciones: Mostrar selección de tema para la partida.

Operación: SeleccionarTema(Tema)

Actores: Usuario, sistema.

Responsabilidades: Caso de uso "Nueva Partida".

Precondiciones: Ninguna.

Postcondiciones: Selección por parte del usuario de un tema para la partida.

Operación: MostrarSeleccionNombre()

Actores: Usuario, sistema.

Responsabilidades: Caso de uso "Nueva Partida".

Precondiciones: Ninguna.

Postcondiciones: Mostrar selección de un nombre para un jugador.

Operación: IntroducirNombre(Nombre)

Actores: Usuario, sistema.

Responsabilidades: Caso de uso "Nueva Partida"

Precondiciones: Ninguna

Postcondiciones: Inserta un nombre de un jugador.

Modelo de comportamiento Partida

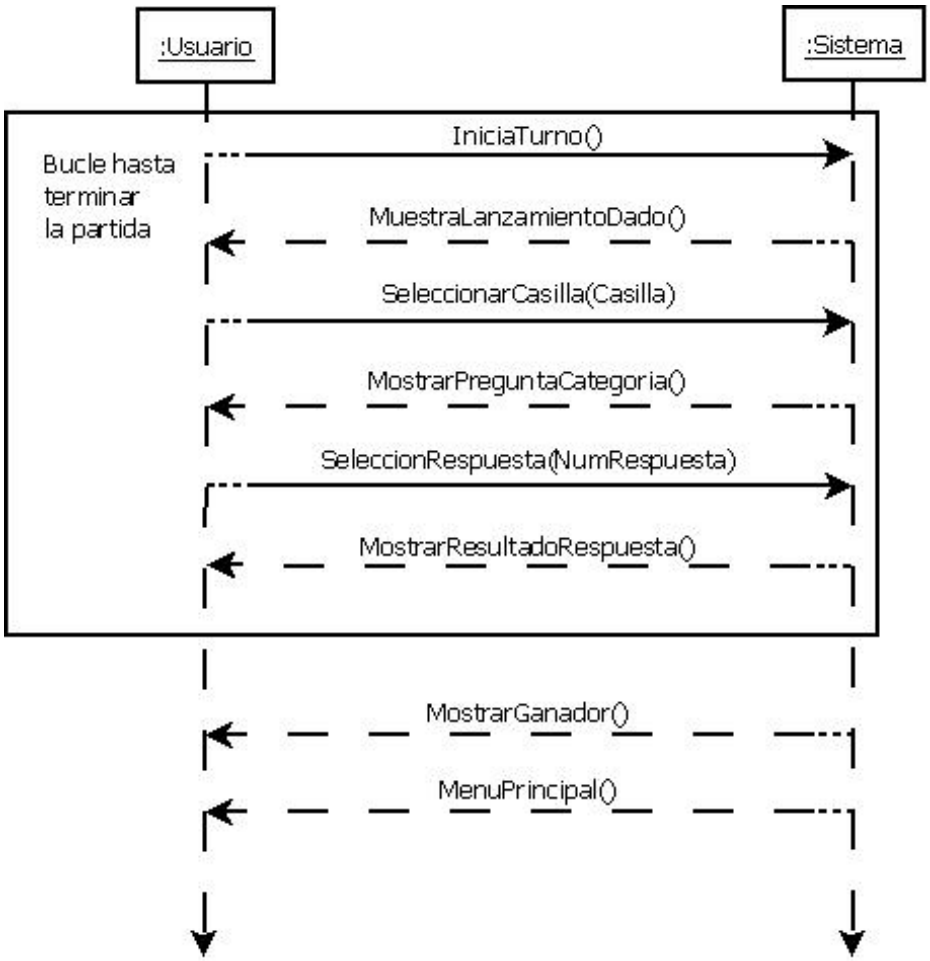


Figura 6.4: Diagrama de secuencia del caso de uso: Partida

Operación: IniciarTurno()

Actores: Usuario, sistema.

Responsabilidades: Caso de uso "Partida".

Precondiciones: Ninguna.

Postcondiciones: Comienzo del turno de un jugador en la partida.

Operación: MuestraLanzamientoDado()

Actores: Usuario, sistema.

Responsabilidades: Caso de uso "Partida".

Precondiciones: El jugador ha iniciado el turno.

Postcondiciones: Mostrar una tirada aleatoria de los dados.

Operación: SeleccionarCasilla(Casilla)

Actores: Usuario, sistema.

Responsabilidades: Caso de uso "Partida".

Precondiciones: Ninguna.

Postcondiciones: Selección por parte del usuario de la casilla a la que quiere mover la ficha.

Operación: MostrarPreguntaCategoria()

Actores: Usuario, sistema.

Responsabilidades: Caso de uso "Partida".

Precondiciones: Jugador seleccionó una casilla que tiene asignada una categoría.

Postcondiciones: Mostrar pregunta que se ha seleccionado aleatoriamente.

Operación: SeleccionRespuesta(NumRespuesta)

Actores: Usuario, sistema.

Responsabilidades: Caso de uso "Partida".

Precondiciones: Ninguna.

Postcondiciones: Selección por parte del usuario de una de las respuestas posibles de la pregunta formulada.

Operación: MostrarResultadoRespuesta()

Actores: Usuario, sistema.

Responsabilidades: Caso de uso "Partida".

Precondiciones: Jugador seleccionó una de las posibles respuestas o termino el tiempo de respuesta.

Postcondiciones: Realizar las acciones pertinentes sobre la puntuación y el estado del jugador dependiendo si la respuesta es correcta o no.

Operación: MostrarGanador()

Actores: Usuario, sistema.

Responsabilidades: Caso de uso "Partida".

Precondiciones: El jugador que tiene el turno ha conseguido la última porción que le quedaba.

Postcondiciones: Se finaliza la partida indicando el jugador ganador y se registra.

Operación: MenuPrincipal()

Actores: Usuario, sistema.

Responsabilidades: Caso de uso "Partida".

Precondiciones: La partida se ha finalizado.

Postcondiciones: Se vuelve al menú principal del juego.

Capítulo 7

Fase de Diseño del Juego

Esta fase de diseño toma los análisis de la etapa anterior y se realiza un diseño exhaustivo de los componentes que componen el juego, para que después en la etapa posterior de implementación abarque todo el proyecto. En esta fase se intervienen los siguientes puntos:

- Diseño de la interfaz del usuario.
- Diseño de los elementos que componen el juego.
- Diseño de la base de datos.
- Estudio de las clases que componen el proyecto.

7.1. Diseño de la interfaz del usuario

La interfaz de usuario es la encargada de interactuar con el usuario y la lógica de la aplicación. Debe mostrar de una manera clara e intuitiva las posibles acciones que puede llevar a cabo el jugador durante el uso del juego. En nuestro caso se va a diseñar dos interfaces de usuario: La primera es la interfaz de usuario del menú principal del juego, donde el jugador puede navegar por las diferentes opciones, y seleccionara las configuraciones para la partida, y la segunda interfaz es la correspondiente a la interfaz del transcurso de una partida que indica, tanto la información del jugador en curso, como las posibilidades disponibles para el jugador durante su turno.

Toda la interfaz está compuesta por tres elementos que se combinan entre sí:

- **Fondos o backgrounds**: Imágenes que ocupan la pantalla o parte de ella que tiene el papel de ser el fondo de la pantalla.
- **Sprites**: Imágenes más pequeñas que representan los objetos de la pantalla.
- **Texto**: Cuando se desea mostrar texto que no se puede adjuntar a las imágenes se necesita de texto plano.

Todas las imágenes están hechas a doscientos cincuenta y seis colores y son en formato BMP, además el color magenta, la consola lo interpreta como transparente al mostrarlo en pantalla.

7.1.1. Interfaz del menú del juego

En esta interfaz se determinara las pantallas necesarias para la configuración de una partida por parte del jugador para una nueva partida, o ver información relacionada del juego.

En la imagen siguiente podemos ver el menú principal del juego donde están las cuatro acciones que puede realizar el jugador cuando ejecuta el juego:

Figura 7.1: Menú principal del juego

La imagen está compuesta por dos fondos que están uno por encima del otro para darle profundidad y movimiento al menú.

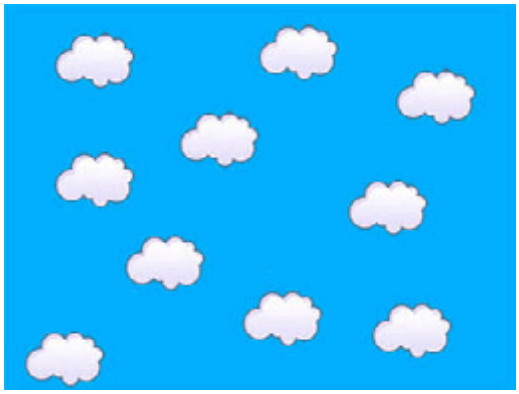


Figura 7.2: Fondo de nubes

Figura 7.3: Fondo del menú

En este menú se puede seleccionar las opciones tanto por el pad de la consola como con el stylus, pulsando en las letras de las opciones que hacen a la vez de botones del menú. En este menú se le ha añadido un fondo móvil para darle más vistosidad mientras se elige las opciones.

En el transcurso de los menús se ha intentado utilizar diferentes imágenes y botones para la navegación rápida entre las opciones del juego como podemos ver en el resto de imágenes que componen el menú, exceptuando un par de pantallas que requieren una complejidad mayor que se explicará más adelante.

Figura 7.4: Menú selector del número de Jugadores

Figura 7.5: Menú selector de tableros

Figura 7.6: Menú selector de Temario

El siguiente menú también pertenece al transcurso de la selección de opciones de una partida, pero a diferencia de las demás es algo más compleja que explicaremos a continuación.

Figura 7.7: Menú de introducción de nombre de los jugadores

En esta pantalla el jugador que selecciona las opciones del menú debe introducir los nombres de todos los jugadores que intervienen en la partida a través del teclado mostrado en pantalla. El teclado está compuesto por celdas de tamaño 8 x 8 que diferencia cada letra de las demás. El funcionamiento del teclado se explicara en la fase de implementación del juego.

Casi todas las pantallas del menú tienen un botón abajo a la derecha para volver al menú anterior si el jugador lo desea.

7.1.2. Interfaz del juego

En la interfaz del juego se mostraran las pantallas correspondientes a lo que se mostrara al jugador durante el transcurso de una partida. La interfaz consta de varias partes que cambian según la pantalla en la que se muestra y el momento en el turno del jugador. Se puede diferenciar dios momentos claves, el momento de la muestra del tablero con las fichas de la partida y el momento que se efectúa la pregunta al jugador para que conteste la pregunta.

Interfaz del tablero

Se muestra la información del jugador actual, como nombre, la puntuación que lleva y las porciones de las categorías que lleva en la pantalla superior. Esto sirve para que el jugador tenga referencia durante este periodo de cómo va su partida y lo que le queda para ganar.

Figura 7.8: Pantalla puntuación jugador

En la pantalla inferior de la consola se muestra la interfaz más importante del juego, ya que muestra el tablero del mismo con las localizaciones de las fichas. El jugador en todo momento sabe su localización y la de los demás para tener un concepto de cómo va la partida.

En esta pantalla se desarrollan las acciones más importantes, como la tirada del dado o la selección por parte del jugador de la casilla a la cual quiere ir según la tirada del dado.

Figura 7.9: Tablero de la partida

Las luces del tablero marcan al jugador las casillas disponibles, una vez se calcula la tirada del dado, a la que puede dirigirse para avanzar por el tablero.

Interfaz de las preguntas

En esta interfaz se muestra la pregunta con las posibles respuestas al jugador para que seleccione la respuesta que considere correcta. En la pantalla superior se formula la pregunta al jugador, mientras que en la pantalla inferior se muestra las tres respuestas posibles junto a un reloj que marcara el tiempo límite para contestar la pregunta.

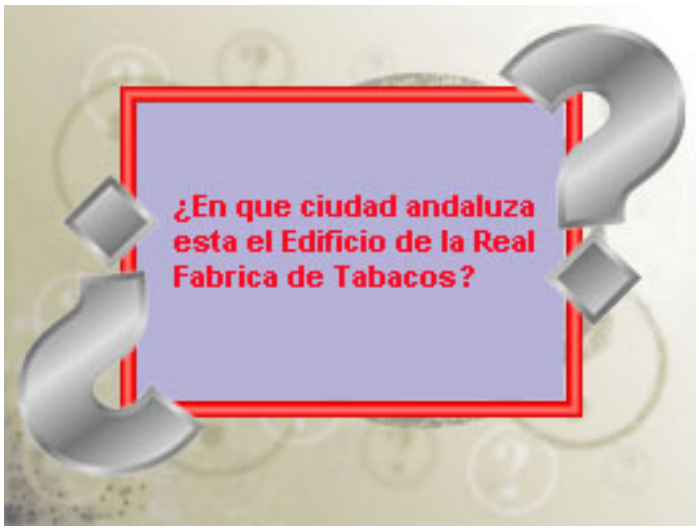


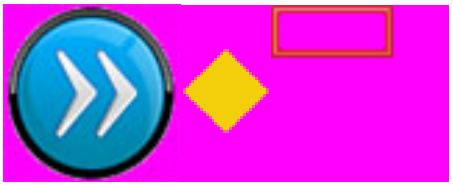
Figura 7.10: Interfaz de la pregunta

Figura 7.11: Interfaz de las respuestas

7.2. Diseño de los elementos que componen el juego.

Los elementos que componen el juego son sprites que representan un elemento real en el juego. Los elementos usados en el juego son:

- **Selectores**: Indican posición en los menús del juego.



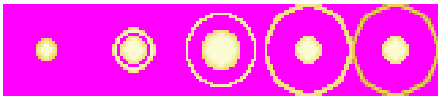
- **Las fichas**: Indican la posición de los jugadores respecto al tablero, además de indicar cuantas porciones de las categorías han conseguido.

- **Porciones de los quesos**: van en conjunto con la ficha.

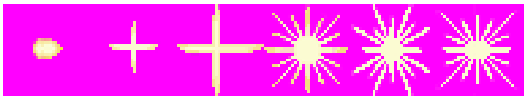
- **El dado**: Indican la tirada de los jugadores al comienzo del turno.

- **El reloj**: Indica el tiempo que te queda para responder una pregunta.

- **Las luces**: Indican las posibilidades que el jugador puede moverse por el tablero.



- **Los fuegos artificiales**: Es un sprite usado al final de una partida que embellece el fin del juego.



- **Flechas**: Indican la respuesta seleccionada por el jugador.

Los últimos cinco sprites son una sucesión de imágenes que emulan el movimiento o la animación del sprite. Cada frame del sprite indica un movimiento determinado que el software de la consola interpreta para animarlo.

7.3. Diseño de la base de datos

En base a las clases usadas en el proyecto se va a proceder al diseño de la base de datos, teniendo en cuenta los datos que se desean almacenar, para agilizar el proceso de carga del juego y para recuperar o guardar información que necesitemos más adelante.

La base de datos se puede dividir en dos partes, ya que no están relacionadas directamente ni indirectamente. Una de las partes son las tablas de almacenamiento del juego, como son los tableros, casillas y temarios del juego. La otra parte corresponde a la información que se guarda de una partida en curso, ya sea una partida guardada por los jugadores, o el ranking de partidas que se rellena según se van realizando partidas.

7.3.1. Base de datos del tablero y temario.

La base de datos del tablero engloba la información de cuatro elementos:

- La información del tablero.
- La información de cada casilla perteneciente al tablero.
- Los puntos cardinales que delimitan cada casilla.
- Los enlaces de las casillas.

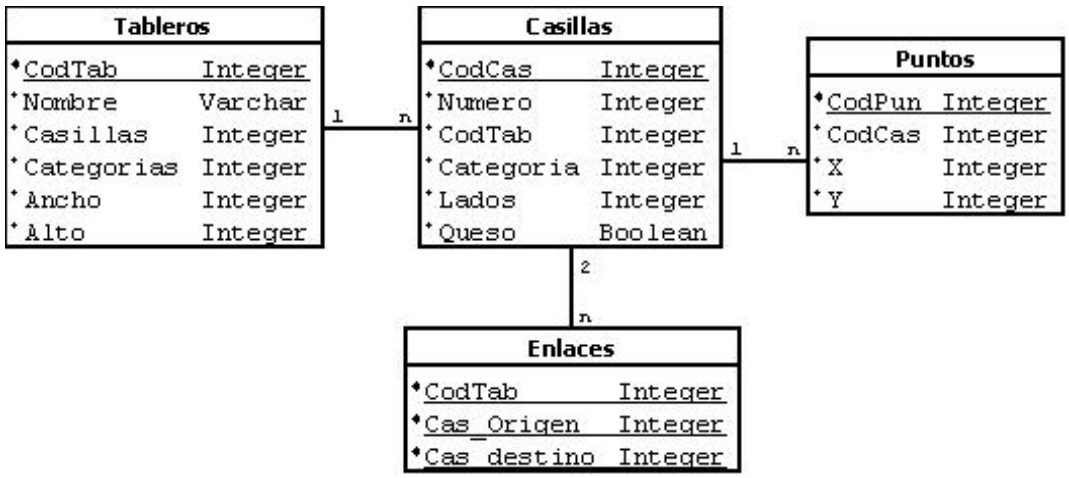


Figura 7.12: Modelo Entidad-Relación del tablero

La información del tablero consta del nombre de los mismos, el número de casillas que contiene el tablero, cuántas categorías puede albergar el tablero, y las medidas de éste, medido en píxeles. Esta información quiere diferenciar cada tablero que se ha agregado al juego. Tanto el número de casillas como el número de categoría se usarán para la carga del contenido del tablero durante el juego, como son las casillas y las categorías del mismo. Las categorías no están enlazadas al tablero porque se eligen

en función al temario que decida el jugador elegir a la hora de crear una nueva partida. Una vez están elegidos tanto el temario como el tablero se escogen tantas categorías del temario como número de categorías soporte el tablero.

La información de las casillas consta del número asignado correspondiente al tablero que pertenece, este dato nos sirve para saber qué número de casilla es según su tablero, la categoría a la que pertenece, que es un número de referencia respecto a las categorías máximas que soporta el tablero, los lados que componen la casilla, para cargar el número de puntos correspondiente al número de lados de la casilla, y un valor booleano que marcara si la casilla es una de las principales respecto al tablero que pertenece.

Los puntos cardinales de las casillas es la información de referencia en la pantalla de cada casilla. Gracias a estas coordenadas podemos saber la localización de la casilla respecto a la imagen del tablero y calcular la posición de la casilla respecto a la pantalla. Las coordenadas X e Y de cada punto están medidos en pixeles tomando como inicio el punto superior izquierda de la imagen del tablero.

Como en todo juego de mesa que tenga un tablero, hay conexiones entre casillas para que el jugador sepa que caminos puede escoger, de los caminos disponibles. La tabla de enlaces indica con su información estos caminos, marcando los enlaces entre casillas que existen para que, después durante el transcurso del juego, se pueda calcular con la tirada del dado a que casillas finales se puede ir por los enlaces. En la tabla se indican los índices de las casillas que están unidas, así como el índice del tablero que pertenece las uniones.

Respecto a las relaciones cabe aclarar que una casilla no puede pertenecer a más de un tablero, que un punto, aunque sea las mismas coordenadas de una casilla a otra, no puede pertenecer a dos casillas, obligando a que cada casilla debe tener tantos puntos como lados tenga. Y respecto a los enlaces, en el mismo tablero no puede haber dos enlaces iguales de un origen a un destino idéntico.

La base de datos del temario consta de la información de tres elementos de la base de datos total:

- La información de los temarios que se puede elegir en el juego.
- La información de las categorías que pertenecen a cada temario.
- La información de las preguntas y respuestas que componen cada categoría.

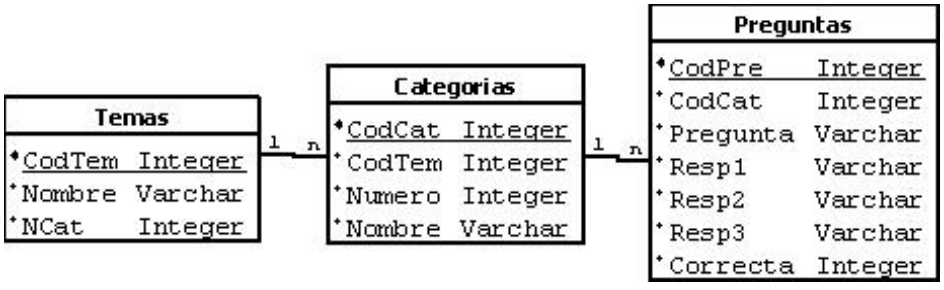


Figura 7.13: Modelo Entidad-Relación del temario

En la tabla de los temas se guarda la información general de los mismos como son el nombre del tema, cual el jugador puede ver durante la selección de temario y es lo que indica al juego que temario

se ha elegido y el número de categorías que compone el temario, que es un número de referencia de cuantas categorías tiene en total.

En la tabla de categorías tendremos la información más individual que compone un tema. Se almacena la información de cada categoría perteneciente a un tema que consta del número de la categoría respecto a su tema que lo diferencia del resto de categorías y el nombre de la categoría para después mostrarlo durante el juego como referencia del mismo.

En la tabla de preguntas ya se encuentra toda la información principal de los temarios, que son las preguntas que componen todas las categorías del mismo. Esta información está compuesta por la pregunta y sus tres respuestas posibles, además de un número que indica cual de las tres respuestas posibles es la correcta.

En cuanto a las relaciones de las tablas podemos definir que una categoría inicialmente no puede pertenecer a más de un tema, para evitar posibles equivocaciones a la hora de modificarlas por parte del usuario. Y las preguntas no pueden pertenecer a más de una categoría por las mismas razones que la categoría.

7.3.2. Base de datos partidas guardadas y ranking

La base de datos de las partidas guardadas consta de:

- La información de la partida guardada.
- La información de los jugadores que están jugando a la partida guardada.

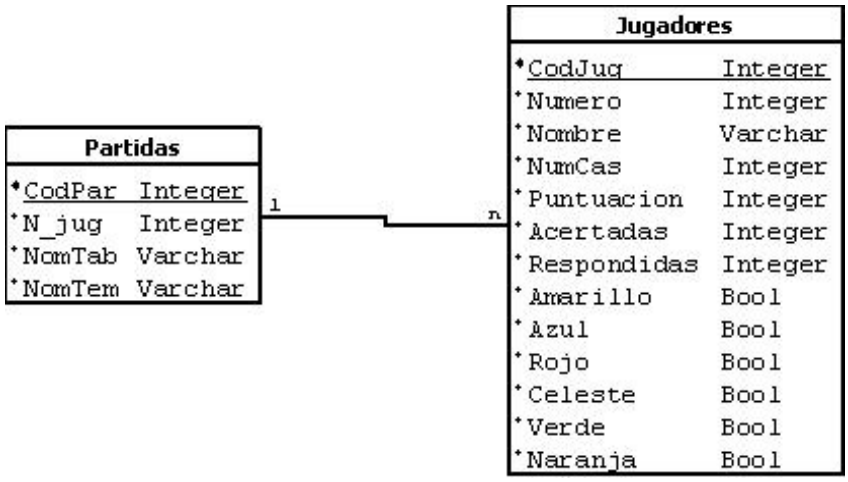


Figura 7.14: Modelo Entidad-Relación de las partidas

En este apartado de la base de datos alberga la información necesaria para la reanudación de una partida que se desea continuar más adelante. Para ello almacenamos la información de la partida en

curso, y la información de todos los jugadores que intervienen en la misma, para una posterior carga de los datos para la continuación de la partida.

Para la partida se guarda el número de jugadores que están participando, así como el nombre del tablero y temarios elegidos por el jugador cuando se creó la partida. Esta información se utilizara para cargar de nuevo en la memoria tanto el tablero como el temario.

Para los jugadores se debe almacenar toda la información generada durante la partida para poder continuar exactamente donde se dejo. Por eso se almacena la información del nombre del jugador en curso, la posición de su ficha respecto al tablero indicando el número de la casilla que se encuentra, la puntuación del jugador actual en el momento de guardar, las preguntas acertadas así como las respondidas totales, para calcular también los fallos que ha cometido el jugador, y por ultimo seis datos que corresponden a las porciones que ha conseguido el jugador obtener de casillas principales del juego, que se indican con un valor verdadero o falso si lo ha conseguido o no.

En esta versión del juego solo se puede guardar una partida a la vez, así que todos los jugadores solo pueden pertenecer a una partida.

Por último está la tabla del ranking que guarda la información de los diez ganadores de partidas con mas puntuación. Toda puntuación que quede por debajo de los diez mejores se eliminara de la tabla.

Figura 7.15: Tabla Ranking

En la tabla se almacena el nombre del ganador, correspondiente al nombre del jugador que se puso a la hora de iniciar la partida y la puntuación obtenida durante la partida.

Capítulo 8

Implementación

8.1. Presentación de cada clase del proyecto

Anteriormente se ha mostrado el diagrama de clases que componen el proyecto. En este apartado se realizara un estudio más detallado de cada clase y sus relaciones, en un modelo basado en programación orientada a objetos. En el siguiente diagrama se describen las clases y objetos de la aplicación y las relaciones que lo componen. Cada clase está compuesta por atributos, que son las propiedades de la clase, y métodos que describen las funcionalidades de las clases.

Para la representación del conjunto de clases, se emplea el estándar UML. Donde una clase se muestra como un rectángulo dividido en tres partes: nombre de la clase, atributos y métodos.

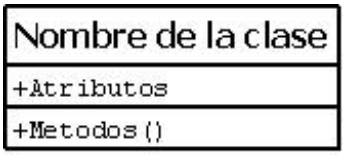


Figura 8.1: Ejemplo de representación de una clase

Se explica en cada clase los métodos de la misma y las pruebas realizadas en la clase.

8.2. Implementación de las clases que componen el proyecto

8.2.1. Clase SQL

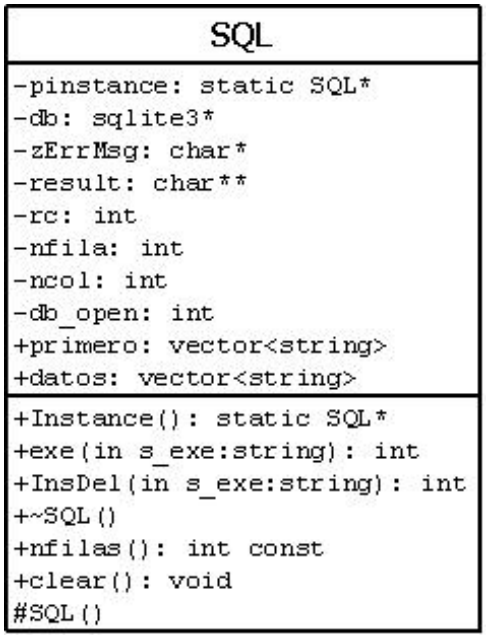


Figura 8.2: Clase SQL

Diseño e implementación

La clase *SQL* es la encargada de la conexión y gestión de la base de datos del juego. Es una clase singleton, es decir, está diseñado para restringir la creación de objetos a uno solamente, este proceso se lleva a cabo a través de un puntero al objeto creado la primera vez que se instancia la clase, que apunta siempre al objeto y está en el ámbito global. Para que la clase sea singleton, el constructor está definido como *protected* para que no se puedan crear objetos desde ninguna parte que no sea la función estática de la clase llamada *Instance*, que realiza la comprobación de existencia del objeto que apunta el puntero, sino existe crea el objeto y si existe devuelve el objeto apuntado por *pinstance*.

Constructor

En el constructor de la clase se indica el nombre y la localización de la base de datos, para ello se necesita la ayuda de la biblioteca libre de libnds[2] llamada *fat.h* que podemos indicar, dentro de la consola, cual es la localización exacta del archivo de la base de datos para que las funciones de Sqlite3[11] puedan encontrarla. Una vez definida la localización de la base de datos, en el caso del juego en la memoria SD de la consola, las funciones de Sqlite3[11] se encargan de cargar la base de datos en el objeto creado de la clase.

La base de datos es un fichero con extensión BD en formato de texto, que la librería interpreta y realiza sentencias SQL sobre ella, donde se puede realizar consultas, inserciones y actualizaciones de la base de datos para su completa gestión.

Métodos

La clase tiene dos métodos principales para realizar cualquier consulta SQL. Tanto al método **exe** como **InsDel** se le pasa por parámetros una cadena de texto con la sentencia SQL a ejecutar sobre la base de datos. La diferencia entre estos dos métodos se alberga en que el método **exe** está orientado a, solamente, consultas sobre la base de datos para recuperar los datos del mismo, y el método **InsDel** está orientado al resto de sentencias SQL que no requieren de respuesta por parte de esta.

En el método **exe** se realiza una lectura de la base de datos en base a la consulta recibida e inserta los datos en dos vectores que son atributos de la clase que se le pasan por referencia. Uno de los vectores hace referencia a los nombres de las columnas leídas en la consulta y el otro vector, es un vector unidimensional con toda la información de las celdas de la tabla consultada. Al ser un vector unidimensional, todas las líneas están correlativas una detrás de otra, así que para la correcta interpretación se puede realizar de una forma sencilla expresada a continuación:

```
for (int i = 0; i < sql->nfilas(); i++){
    x = sql->datos[(i*4)+3];
}
```

Donde según el número de líneas que contenga la tabla, que también se devuelve al realizar la consulta, se recorre el vector multiplicando el contador por el número de columnas que contenga la tabla, mas el número correspondiente a la localización de la columna respecto a la primera, donde la primera columna seria el número cero, la segunda columna sería el número uno y así sucesivamente, la función sería algo así: "(contador x columnas) + localización".

Pruebas

La batería de pruebas realizadas a la clase se basan en la ejecución de todas las diferentes consultas SQL posibles a través de sqlite3[11] como son sentencias SELECT, INSERT, UPDATE y DELETE, y la comprobación de los resultados. Como no es necesaria la creación ni la eliminación de tablas por parte del juego, no se han realizado pruebas para ello. La batería también engloba las pruebas por ser clase singleton, para comprobar que no se pueden crear más de un objeto de la clase de ninguna manera además de que se realiza una referencia correcta al objeto creado la primera vez que se construye el objeto.

8.2.2. Clase Matriz

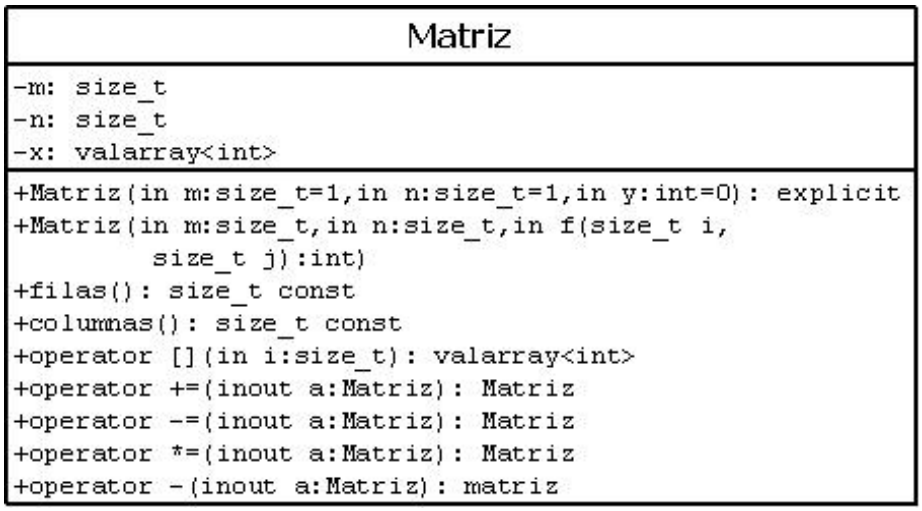


Figura 8.3: Clase Matriz

Diseño e implementación

La clase Matriz es utilizada para la simulación de una matriz matemática para realizar los cálculos correspondientes a ellas. En el proyecto se ha utilizado la matriz para crear una matriz de adyacencia para indicar, entre todas las casillas pertenecientes a un tablero, que casillas están conectadas entre sí para indicar al juego los caminos posibles que puede recorrer una ficha, con el número de una tirada y su posición actual.

Constructores

La clase tiene dos constructores especificados para la creación de una matriz. Un constructor donde se le indica las dimensiones de la matriz tanto en filas y columnas y un valor predeterminado para las casillas de la matriz, y otro constructor donde los valores de las celdas de la matriz viene determinado por una función descrita anteriormente.

Métodos

Contiene métodos de observación de las columnas y filas que contiene la matriz creada y también tiene definidas las operaciones básicas que se pueden realizar con las matrices, matemáticamente.

Pruebas

A la clase Matriz se le han realizado pruebas básicas de cálculo de matrices, así como, la creación y destrucción de las matrices creadas.

8.2.3. Clase Categoría

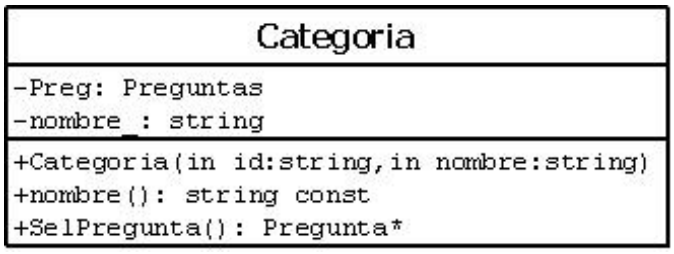


Figura 8.4: Clase Categoría

Diseño e implementación

Los temas no tienen clase definida, ya que por la poca información que se tiene de ellos, no es necesario implementarla, en cambio, la clase *Categoría* representa cada categoría que pertenece a un temario definido en la base de datos. Una categoría se diferencia de las demás por su nombre y por el temario al que pertenece y es la encargada de seleccionar una de las preguntas de las numerosas preguntas que contiene la misma. Así que cada categoría contendrá una lista completa de todas las posibles preguntas que pertenecen a ella para su acceso.

Para la lista de preguntas se ha implementado un map de objetos de la clase Pregunta, junto con un valor booleano, ordenados por el índice de la pregunta, este vector es cargado desde la base de datos tomando como referencia el código de la categoría *CodCat* a las que pertenecen. Los valores booleanos enlazados a cada pregunta por el map, indica si la pregunta en cuestión, ha sido seleccionada anteriormente en el transcurso del juego, para que no se vuelva a repetir la pregunta. Esto solo se conservará si es la misma partida.

Constructor

En el constructor de la clase Categoría se leen de la base de datos toda la información respecto a la categoría y las preguntas que pertenecen a la categoría, creando un objeto por cada pregunta e insertándolo en el vector antes comentado. Cuando se crea un objeto casilla se leen los textos de las preguntas de la base de datos.

Métodos

La función **SelPregunta** es una de las más importantes del juego, ya que es la encargada de la selección aleatoria de una de las preguntas que pertenecen a la categoría que ejecuta el método.

```
Pregunta* Categoria::SelPregunta(){
    u32 rand, max;
    Pregunta *P;
```

```
        bool bucle = false;
        Preguntas::const_iterator i;
        vector<Pregunta*> Seleccion;
        while (bucle == false){
            for (i = Preg.begin(); i != Preg.end(); i++){
                if ((*i).second == false)
                    Seleccion.push_back((*i).first);
            }

            if(Seleccion.size() > 0){
                max = Seleccion.size();
                rand = PA_RandMax(max); //Genera número aleatorio entre 0 y max
                P = Seleccion[rand];
                bucle = true;
                Preg[P] = true;
            }else{
                for (i = Preg.begin(); i != Preg.end(); i++)
                    Preg[(*i).first] = false;
            }
        }
        return P;
    }
```

Para la selección de la pregunta aleatoria para el juego, se seleccionan primero las preguntas que no han sido seleccionadas anteriormente en la partida, agrupándolas en un vector único de preguntas. Posteriormente Se genera un número aleatorio sirviéndonos de la función proporcionada por la biblioteca PAlib[6] llamada **PA_Rand** que genera un número aleatorio comprendido entre el cero y un número máximo, que en nuestro caso es el número de preguntas sin seleccionar. Una vez escogido el número aleatorio, se cambia el valor booleano de la pregunta escogida a verdadero, para que no vuelva a escogerse en un futuro y se devuelve el objeto de la clase *Pregunta* con la información que se necesita para su formulación.

Si en la selección de preguntas sin elegir, no hay ninguna pregunta, ya que todas han sido formuladas, se reinicia todos los valores booleanos a falso de todas las preguntas, para que se puedan repetir las preguntas.

Pruebas

En las pruebas de la clase se ha creado varias categorías definidas en la base de datos para comprobar que se cargaba correctamente el map de preguntas, y se ha comprobado que en la función de selección de preguntas no devuelve ningún valor nulo aunque todas las preguntas hayan sido seleccionadas.

8.2.4. Clase Pregunta

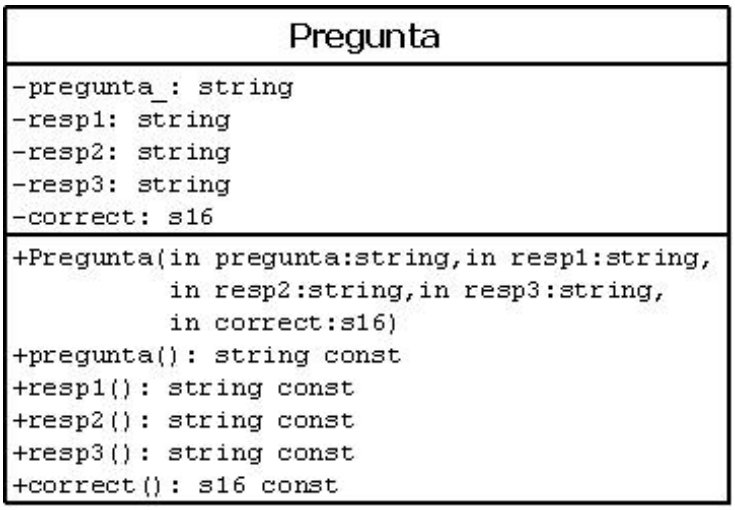


Figura 8.5: Clase Pregunta

Diseño e implementación

Esta clase es una representación de una de las preguntas perteneciente a una categoría. Cada pregunta almacenara la información correspondiente del texto de la pregunta, así como el texto de cada respuesta posible a la pregunta y un valor numérico que indica cual de las tres respuestas es la correcta. La clase solamente dispone de funciones de consulta de los textos que contiene para la muestra por pantalla de los mismos.

Constructor

El constructor de la clase recibe la información correspondiente a la pregunta, respuestas y la respuesta correcta para su registro en el sistema y almacenamiento.

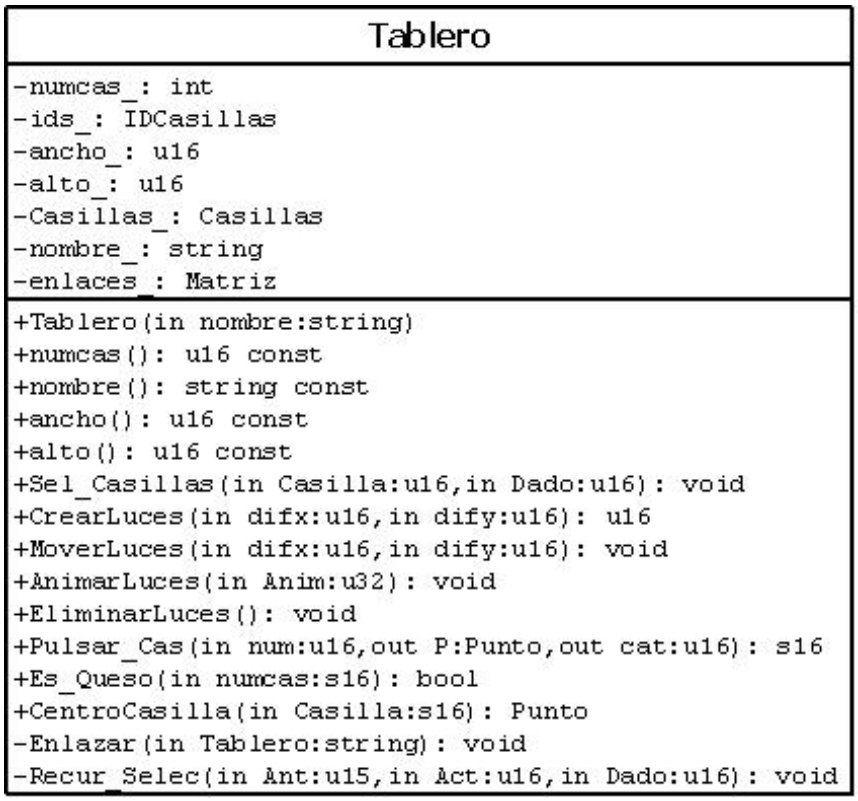
Métodos

Las funciones de consulta corresponden a las tres respuestas posibles a elegir durante el juego, el texto de la pregunta, y el número entero correspondiente a la respuesta correcta de la pregunta.

Pruebas

Para las pruebas de esta clase simplemente se han creado objetos de la misma y comprobado que se devuelve correctamente los valores de las cadenas de texto de la pregunta.

8.2.5. Clase Tablero



Diseño e implementación

La clase especificada actual, corresponde a la representación del objeto perteneciente al tablero del juego. Esta clase se encarga de gestionar, tanto la creación y eliminación del tablero, sino que además, gestiona los movimientos de las fichas por la pantalla según se mueva el tablero y el cálculo del movimiento de las casillas según la tirada efectuada con el dado con la creación de sprites de tipo luces, que se colocan sobre las casillas que un jugador puede optar para moverse.

Esta clase tiene como parámetros públicos dos vectores importantes:

- **IDCasillas**: Corresponde a los identificadores de las casillas que pertenecen al tablero seleccionados durante el recorrido del mismo que indican las casillas que el jugador puede optar cuando es su turno de mover y se sabe la tirada del dato del turno.
- **Casillas**: Es un vector de objetos de clase casilla, ordenados por el identificador de la casilla, para un mejor acceso a la información correspondiente a las casillas que pertenecen al tablero.

Constructor

En el constructor del tablero, además de la carga desde la base de datos de la información correspondiente al tablero elegido durante el juego, como es la información de tamaño de la imagen del tablero, el número de casillas correspondiente y el nombre, se realizan dos tareas importantes para el proceso de carga del juego. Uno de los procesos se encarga de la carga de toda la información perteneciente a todas las casillas que pertenecen al tablero y el otro proceso es el enlace de dichas casillas entre si, a través de la creación de una matriz de adyacencia, que indica con cero o uno si las casillas son contiguas o no. Durante la carga de las casillas se va comprobando desde la base de datos si la casilla actual es una de las casillas principales que proporcionan una porción del queso del juego, esta información se registra en la casilla para consultarla posteriormente.

El proceso de enlace de las casillas simplemente se realiza un recorrido por la tabla de enlaces que pertenecen al tablero, y los índices que están marcados en la base de datos, que pertenecen a casillas, se marcan con un uno las celdas, donde la casilla origen corresponde a la fila de la matriz, mientras que el destino corresponde a la columna y viceversa, ya que por el tablero se puede recorrer en ambos sentidos. Por ejemplo siendo *Enlaces* la matriz de enlaces de las casillas, si tenemos que están enlazadas la casilla con identificador número dos y la casilla con el número cuatro, entonces registraremos en la matriz, **Enlaces[origen][destino] = 1** y **Enlaces[destino][origen] = 1**.

Métodos

El método público llamado **Sel_Casillas**, es uno de los métodos más importantes del juego, ya que es la encarga de seleccionar las casillas que se deben marcar durante el juego cuando un jugador landa un dado. Este método llama a su vez a otro método que se va a encargar de la ejecución recursiva para recorrer el tablero tantas casillas como las marcadas por los dados, y registrar en el vector "IDCasillas" el identificador de la casilla que es posible seleccionar. Para este proceso el método principal recibe el identificador de la casilla que se encuentra actualmente la ficha del jugador y el número de la tirada del dado, que corresponde a la suma de los números de los dos dados. Como es imposible sacar el número uno en los dados, ya que se lanzan dos dados y como mínimo es dos, el primer paso lo realiza el método principal dando después el control al método privado recursivo para el proceso posterior.

Para la realización de la recursividad por el tablero recorriendo las casillas, se reciben dos identificadores de las casillas, uno de los identificadores corresponde a la localización virtual actual que se encuentra la ficha, y el identificador de la casilla de la localización anterior que estaba la ficha, se necesita el identificador anterior para conocer el camino escogido por la recursividad (ya que los caminos de las casillas solo tienen dos direcciones) para que la ficha no vuelva atrás si ha escogido una dirección. Cada vez que se avanza una casilla se reduce en uno el valor del dado proporcionado, hasta que el dado llega a cero. Si el dado llega a cero quiere decir que la casilla que se encuentra actualmente la ficha, virtualmente, es una de las casillas que puede optar el jugador para moverse, y se añade el identificador al vector de selección, si el dado no es cero, entonces se llama de nuevo a la recursividad indicando la casilla actual como la casilla anterior, y la casilla seleccionada a moverse, comprobando su enlace en la matriz de adyacencia, correspondería ahora a la casilla actual. En la selección de la casilla a avanzar si la casilla es la casilla anterior por donde viene se descarta.

```
void Tablero::Sel_Casillas(u16 Casilla, u16 Dado){
    ids_.clear();
```

```
    for(int i = 0; i < numcas_; i++){
        if(enlaces_[Casilla][i] == 1)
                Recur_Selec(Casilla, i, Dado - 1);
        }
}
void Tablero::Recur_Selec(u16 Ant, u16 Act, u16 Dado){
        if(Dado == 0)
            ids_.push_back(Act);
        else{
            for(u16 i = 0; i < numcas_; i++){
                if(enlaces_[Act][i] == 1 && Ant != i)
                    Recur_Selec(Act, i, Dado - 1);
            }
        }
}
```

Existen cuatro métodos para indicar al jugador las casillas a las que puede moverse cuando se calcula dependiendo de su tirada. Uno de los métodos es **CrearLuces**, otro corresponde a **MoverLuces**, otro a **AnimarLuces** y por ultimo **EliminarLuces**.

- **CrearLuces:** es el encargado de recorrer el vector de identificadores de las casillas elegidas por el método recursivo, calcular el dentro de la casilla y colocar un sprite del tipo "Luz" en la localización marcada del centro.
- **MoverLuces:** es el encargado de mover las luces por la pantalla, conservando su posición en el tablero, cuando el jugador desea mover la pantalla por el tablero.
- **AnimarLuces:** es un método que se está ejecutando continuamente durante el transcurso de la selección de la casilla por el jugador, que se encarga de la animación de las luces creadas anteriormente. Esta animación se realiza a través del método de PAlib[6] llamado *PA_SetSpriteAnim* que, indicándole el número del sprite correspondiente a la luz, la pantalla donde se encuentra la luz y el número de la progresión de la animación, anima el sprite cargando el fragmento de la imagen correspondiente. Este proceso se realiza en todas las luces creadas.
- **EliminarLuces:** simplemente elimina todos los sprites de la pantalla una vez que el jugador ha seleccionado una casilla, o en su defecto ha ejecutado el menú de pausa o la ayuda de las categorías.

Para el movimiento de las luces y fichas, ya que algunas luces pueden no estar en el rango de la pantalla puesto que la pantalla es más pequeña que el tablero, cada vez que se mueve la pantalla por el tablero, se va incrementando o reduciendo dos valores enteros que indican la distancia en pixeles que se encuentra la pantalla respecto a la esquina superior izquierda del tablero, con los valores de posición de la pantalla se realiza un cálculo donde se restan la posición del sprite en el tablero, a la posición de

la pantalla respecto al tablero. Como estar eliminando y creando sprites continuamente sobrecarga la memoria de la consola, si el valor de la resta determina que el sprite se encuentra fuera de la pantalla se quedara en el borde de la pantalla, sino pues el sprite de moverá por la pantalla conservando su posición en el tablero.

```
x = p.x() - difx; //p es la coordenada en el tablero y difx es la distancia de la pantalla
en el eje x respecto al tablero.
y = p.y() - dify; //p es la coordenada en el tablero y dify es la distancia de la pantalla
en el eje y respecto al tablero.
if(x < 0)
    x = 0;
else if(x > 256)
    x = 256;
if(y < 0)
    y = 0;
else if(y > 192)
    y = 192;
```

El último método del tablero corresponde al método de selección por parte del jugador de la casilla a la que desea mover su ficha, de entre las opciones proporcionadas posibles para su movimiento, determinado por la tirada del dado. Este método devuelve la posición del dentro de la casilla seleccionada, así como la categoría de la misma, para la selección de la pregunta, por referencia y el identificador de la casilla para actualizarlo en el objeto del jugador.

Pruebas

Para las pruebas de esta clase se ha creado un objeto del tablero básico del juego y comprobado por salida estándar, que la carga de las casillas y enlaces se ha realizado correctamente. Para la comprobación del método de selección de casillas se ha simulado el transcurso de una partida con valores por defectos para comprobar que las casillas seleccionadas corresponden a las que deberían ser según el dibujo del tablero, marcando en el dibujo los números de los identificadores que tienen las casillas. Y por ultimo en el movimiento de las luces se ha mostrado por pantalla tanto el tablero como una serie de sprites y pulsando las teclas de dirección de la consola he podido comprobar el movimiento de las luces.

Diseño e implementación

La clase punto es una clase básica que representa la coordenada X e Y del tablero o de la pantalla para una mejor utilización y facilidad de reutilización durante el juego.

Constructor

El constructor de la clase simplemente recibe dos números enteros correspondientes a la coordenada X y la coordenada Y en pixeles de un punto.

Métodos

Esta clase solo tiene métodos de observación de las coordenadas que contiene.

Pruebas

Esta clase es bastante simple en su implementación, así que simplemente se han realizado pruebas simples de creación de objetos de esta clase para su utilización en varias funciones de cálculo de coordenadas.

8.2.6. Clase Casilla

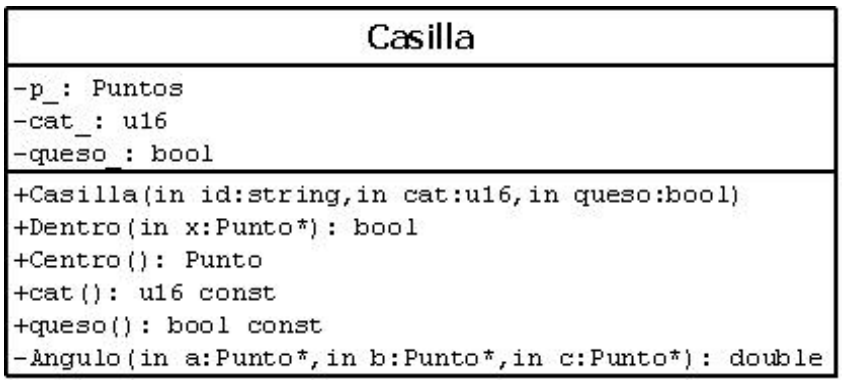


Figura 8.6: Clase Casilla

Diseño e implementación

La clase Casilla corresponde a la representación de cada una de las casillas que pertenecen a un tablero, delimitado por líneas, donde un jugador puede mover su ficha. Estas casillas son utilizadas para que el jugador se pueda mover a través del tablero e indicar el tipo de pregunta que se debe realizar al jugador.

Esta clase tiene definido un vector privado de objetos de la clase *Punto* que determinan virtualmente las delimitaciones de la casilla respecto al tablero, para realizar cálculos de posición.

Constructor

En el constructor de la clase se recibe el identificador correspondiente a la casilla a cargar desde la base de datos, donde se extraen los datos de la categoría que pertenece y si es una de las casillas principales del tablero. Este método recorre la tabla de puntos pertenecientes a la casilla y registra estos puntos en el vector de puntos.

Métodos

La clase contiene un método encargado del cálculo del centro de la casilla, teniendo en cuenta los puntos que delimitan a la casilla. Para ello se realiza la suma de todas las coordenadas X por un lado y todas las coordenadas Y por otro y se calcula la media de las dos sumas. Los resultados de las medias es un punto que corresponde al centro geométrico de la casilla.

Además contiene dos métodos que no se llegan a utilizar pero que en un futuro pueden ser útiles. Estos dos métodos se encargan de calcular si, dado un punto, este punto se encuentra o no dentro de la casilla a comprobar. Este punto es proporcionado normalmente por el Stylus, ya que se quiere comprobar si se ha pulsado una casilla o no. Para la comprobación del proceso, se realiza el cálculo matemático del ángulo que forman el punto proporcionado con todos los vértices de la casilla tomados de dos en dos, posteriormente se suman todos los ángulos proporcionados para obtener el ángulo total. Si este ángulo total es muy cercano a cero o cero, Se determina que los ángulos se han anulado y por consiguiente el punto está dentro de la casilla, en cambio si el resultado es mayor o menos que cero se determina que el punto está fuera de la casilla. Estos métodos corresponden a **bool Dentro(Punto* x)** y **double Angulo(Punto* a, Punto* b, Punto* c)**.

Pruebas

Para las pruebas de esta clase se han realizado junto a las pruebas del tablero donde se comprueba si la carga de los datos de las casillas y los puntos que las componen son correctos, y visualmente se ha comprobado que el método de cálculo del centro se realiza correctamente respecto a una casilla seleccionada aleatoriamente.

8.2.7. Clase Ficha

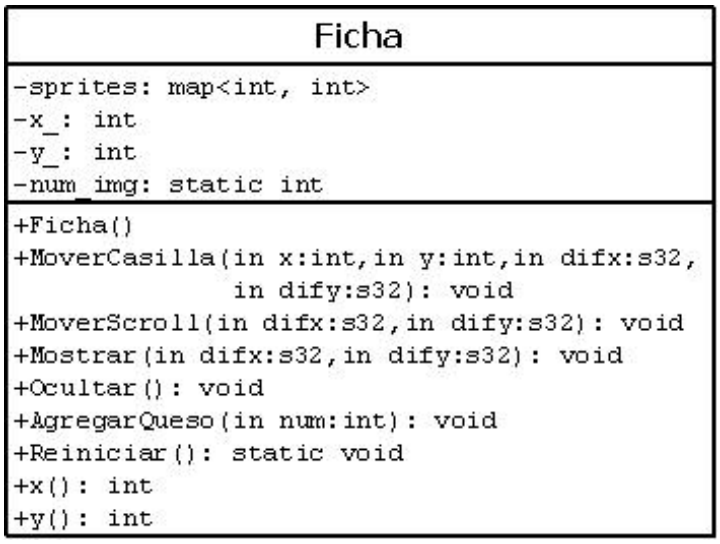


Figura 8.7: Clase Ficha

Diseño e implementación

La clase *Ficha* representa la ficha del juego que tiene el jugador en una partida. Cada ficha pertenece a un único jugador que determina la posición actual del jugador respecto al tablero, así como el progreso que lleva el jugador respecto a los quesos conseguidos al caer en casillas principales del tablero.

Para la creación del sprite de la ficha se utiliza una variable estática que determina el número del sprite que corresponde la misma. Este número incrementa cada vez que se agrega un jugador a la partida o se agrega una porción del queso al responder correctamente a una pregunta de una casilla principal y se reinicia cada vez que se comienza o se carga una partida. Esta variable es usada para tener identificados, y a la vez no se repita, el número de los sprites de la pantalla inferior que pertenecen a cada ficha. Cada ficha tendrá como mínimo el sprite principal que corresponde a la ficha simple y podrán añadirse seis sprites más correspondientes a los quesos que pueden conseguir como máximo el jugador.

La ficha tiene un *map* privado que es la relación que tiene los números de los sprites proporcionado por la variable estática y un número que indica que corresponde el número del sprite, ya sea la ficha o las porciones.

Constructor

En el constructor de la clase se carga en memoria y en la pantalla la ficha que se determina el color con respecto a la variable estática se encuentre, ya que inicialmente se cargan los sprites de las fichas de los jugadores que van a participar en la partida. Una vez determinado el sprite principal de la ficha se registra en el *map* la relación del número de la variable estática con el tipo de sprite que

corresponde, en este caso cero, que indica que es la ficha. Por último se decrementa la variable estática, para que ninguna ficha creada posteriormente o porción, pueda usar el número del sprite. Los sprites se cargaran en la pantalla inferior.

Métodos

El método **MoverScroll** es un método ejecutado durante la partida cada vez que un jugador mueve la pantalla por el tablero. Este método recibe las distancias de la pantalla respecto al tablero y, junto a las coordenadas que se encuentran actualmente la ficha, se calcula la posición de la ficha en la pantalla. Si el cálculo se encuentra fuera de la pantalla, el sprite se cargara en el borde de la pantalla, en la dirección que se encuentra la ficha.

El método **MoverCasilla** simplemente cambia los valores de localización de la ficha en el tablero y vuelve a cargar la posición de la misma en la pantalla.

Esta clase tiene dos métodos de mostrar y ocultar la ficha en pantalla según la necesidad. El método **mostrar** recorre el *map* de la ficha a mostrar cargando cada sprite en pantalla como ítems tenga el *map*. El tipo de imagen que será cargada viene determinado por el segundo valor del *map* que es la clase de imagen que es. El color de la ficha viene determinado por el número del sprite que esta enlazado al ítem. En cambio el método **ocultar** simplemente elimina en pantalla todos los sprites que pertenecen al *map* y así no molesten posteriormente.

Por último está el método **AgregarQueso** que recibe un número entero correspondiente a la categoría del queso a agregar, que determina el color de la porción, crea el sprite en pantalla en la misma localización de la ficha y agrega la imagen al *map* de sprites de la ficha.

Pruebas

Con esta clase durante el transcurso del diseño del juego se han creado varios objetos en pantalla simulando las casillas, realizando movimientos con los sprites correspondientes a las casillas y por ultimo comprobando que toda la información, sobre todo la variable estática y el método estático, estén correctamente, tanto en funcionalidad como en los valores que adquiere, para así evitar problemas de duplicación de índices de sprites de las imágenes que se van cargando en el sistema de la consola.

8.2.8. Clase Queso

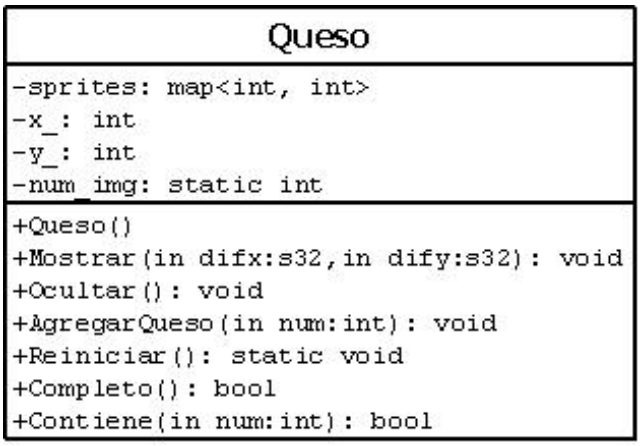


Figura 8.8: Clase Queso

Diseño e implementación

La clase *Queso* es un marcador que se muestra en la pantalla superior de la consola, indicando el progreso del jugador en la partida. No es más que la copia de la ficha en mayor tamaño con las porciones de queso conseguidas.

Igual que la clase *Ficha* para la creación del sprite de la ficha se utiliza una variable estática que determina el número del sprite que corresponde la misma. Este número incrementa cada vez que se agrega un jugador a la partida o se agrega una porción del queso al responder correctamente a una pregunta de una casilla principal y se reinicia cada vez que se comienza o se carga una partida. Esta variable es usada para tener identificados, y a la vez no se repita, el número de los sprites de la pantalla inferior que pertenecen a cada ficha. Cada ficha tendrá como mínimo el sprite principal que corresponde a la ficha simple y podrán añadirse seis sprites más correspondientes a los quesos que pueden conseguir como máximo el jugador.

Constructor

En el constructor de la clase se carga en memoria y en la pantalla la ficha que se determina el color con respecto a la variable estática se encuentre, ya que inicialmente se cargan los sprites de las fichas de los jugadores que van a participar en la partida. Una vez determinado el sprite principal de la ficha se registra en el *map* la relación del número de la variable estática con el tipo de sprite que corresponde, en este caso cero, que indica que es la ficha. Por último se decrementa la variable estática, para que ninguna ficha creada posteriormente o porción, pueda usar el número del sprite. Los sprites se cargaran en la pantalla superior.

Métodos

Esta clase tiene dos métodos de mostrar y ocultar la ficha en pantalla según la necesidad. El método mostrar recorre el *map* de la ficha a mostrar cargando cada sprite en pantalla como ítems tenga el *map*. El tipo de imagen que será cargada viene determinado por el segundo valor del *map* que es la clase de imagen que es. El color de la ficha viene determinado por el número del sprite que esta enlazado al ítem. En cambio el método ocultar simplemente elimina en pantalla todos los sprites que pertenecen al *map* y así no molesten posteriormente.

Por último la clase tiene dos métodos de comprobación que se utilizan para determinar el estado del jugador:

- **Completo**: Este método devuelve *True* si el jugador ha conseguido las seis porciones de queso del juego, ganando así la partida y *False* si aún le queda alguna porción por conseguir.
- **Contiene**: Este método devuelve *True* si el número recibido, correspondiente al color de la porción, se ha conseguido ya por el jugador o no. Este método es usado cada vez que un jugador avanza a una casilla principal del tablero y responde correctamente a la pregunta. Entonces si el jugador no ha conseguido la porción del queso correspondiente al color de la casilla que se encuentra actualmente, se añade el sprite al objeto.

Pruebas

Con esta clase se ha realizado las mismas acciones que se han realizado con la clase ficha, con la única salvedad que las pruebas de movimiento de los sprites correspondientes a esta clase no han tenido que ser comprobados, ya que no requieren ser movidos de la localización que se le otorga inicialmente. Otra diferencia que tiene esta clase respecto a la clase ficha, es que se ha comprobado que los métodos de **contiene** y **completo** devuelve correctamente los valores que deben según va cambiando las porciones que tiene el queso.

8.2.9. Clase Fuego

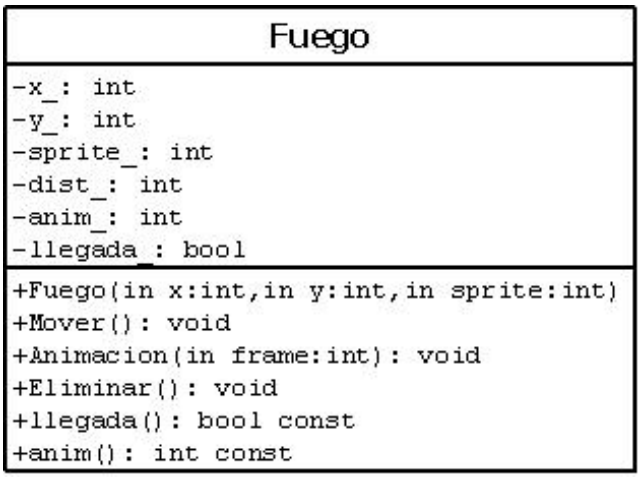


Figura 8.9: Clase Fuego

Diseño e implementación

La clase *Fuego* representa cada uno de los fuegos artificiales que se crean al finalizar la partida. Esta clase se ha añadido para que resulte más fácil la gestión de los mismos durante la ejecución de la pantalla de finalizar partida.

Esta clase tiene ciertas variables privadas que son indicadores de estado del fuego artificial. Uno es el indicador de la distancia recorrida en vertical del fuego respecto a la parte inferior de la pantalla, otro indica si el fuego ha llegado a la distancia donde debería estallar y por ultimo un número que indica la animación actual que se encuentra el fuego.

Constructor

En el constructor de la clase simplemente se indican para su registro las coordenadas que comienza a aparecer el fuego y el número del sprite que corresponde el fuego creado para hacer referencia al mismo para su movimiento y animación.

Métodos

Los métodos públicos de esta clase corresponden a la animación del sprite y su movimiento:

- El método **Mover** mueve verticalmente el sprite por la pantalla la distancia determinada por el valor recibido en el constructor. Si la distancia recorrida es la misma que la distancia que debería recorrer, entonces se indica en el objeto que ha llegado a su destino en la pantalla para realizar la animación. Si no ha llegado simplemente se mueve el sprite. Si el sprite supera el alto de la

pantalla inferior, ya que el número aleatorio de distancia a recorrer puede ser mayor, el sprite se elimina de la pantalla inferior para mostrarse en la pantalla superior en la misma posición X, continuando su recorrido.

- El método **Animacion** recibe el número de frames que se ha recorrido desde que empezó la pantalla de finalización, y calcula para que cada diez frames de pantalla se modifique la animación del sprite.
- El método **Eliminar** simplemente elimina de la pantalla el sprite.

Pruebas

Para esta clase, que solamente se usa en un apartado del juego como embellecedor del escenario, se ha simulado una pantalla en negro donde se van creando objetos de esta clase y se van avanzando en sus animaciones hasta que se completa.

8.2.10. Clase Jugador

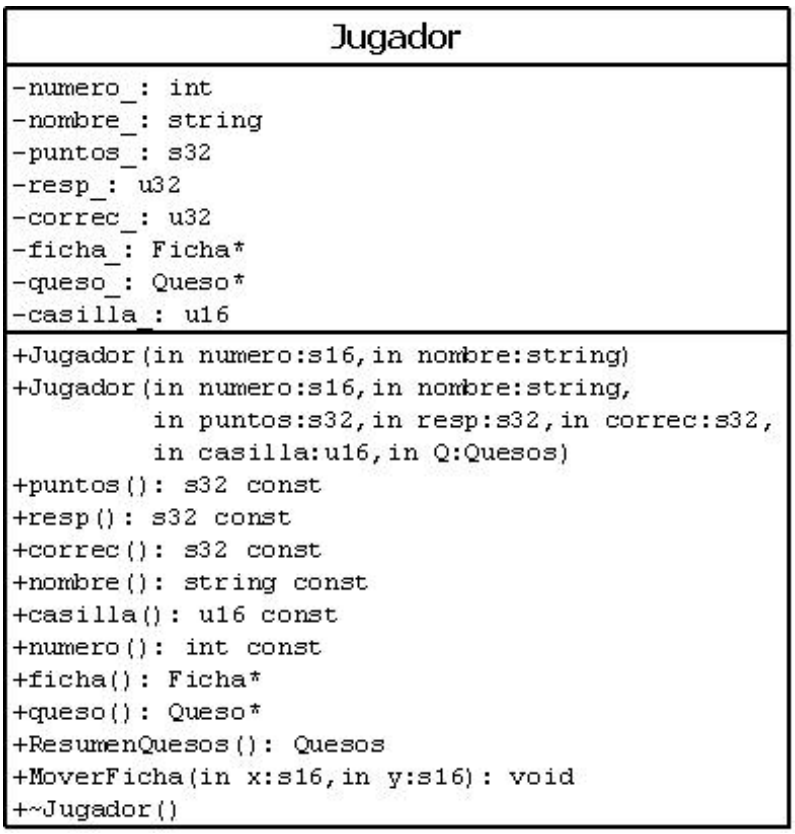


Figura 8.10: Clase Jugador

Diseño e implementación

Esta clase es una de las más importantes del juego, ya que representa a un jugador que está jugando una partida en el juego. Esta clase es la encargada de gestionar toda la información del jugador, ya sea puntuación, objetos que pertenecen al jugador y el progreso del jugador durante la partida.

La clase tiene dos objetos relacionados que corresponden a la ficha del jugador de la clase *Ficha* con la que se moverá por el tablero y un objeto de la clase *Queso* que indicara el progreso del jugador en la partida.

Constructores

Esta clase tiene dos constructores definidos según el tipo de partida se esté ejecutando:

- **Jugador(s16 número, string nombre)**

  Este constructor se utiliza cada vez que se inicie una nueva partida en el juego. Este constructor iniciara todos los valores del jugador a su valor inicial, genera la ficha y el queso del jugador.

- **Jugador(s16 número, string nombre, s32 puntos, s32 resp, s32 correc, u16 casilla, Quesos Q)**

  Este constructor se utiliza cuando la partida que se va a jugar es cargada desde la base de datos como una partida que se guardo anteriormente y se desea continuar. Por consiguiente los valores del jugador se actualizan con los valores recibidos en el constructor y como el jugador podría tener algún progreso respecto a las porciones conseguidas por el mismo, se añadirán los sprites tanto a la ficha como al queso para que prosiga su partida.

Métodos

Además de todos los métodos observadores de la clase para la consulta de todos los datos del jugador tenemos dos métodos públicos que se utilizan durante la partida:

- **Quesos ResumenQuesos(void)**:

  Devuelve el estado de cada uno de las porciones del jugador, devolviendo un vector de números enteros, donde el número uno corresponde que el jugador ha conseguido esa porción y el número cero si el jugador no la ha conseguido.

- **Void MoverFicha(s16 x, s16 y)**:

  Modifica la posición de la ficha del jugador con nuevas coordenadas.

Pruebas

Como la clase *Jugador* es de las más importantes del juego se ha realizado pruebas especificas en el. Primero la creación especifica tanto por el constructor simple, como el completo para comprobar que se crean los objetos correctamente. Se ha realizado pruebas en el objeto ficha del jugador y queso, para comprobar que los objetos están ensamblados correctamente al objeto Jugador.

8.2.11. Clase Partida

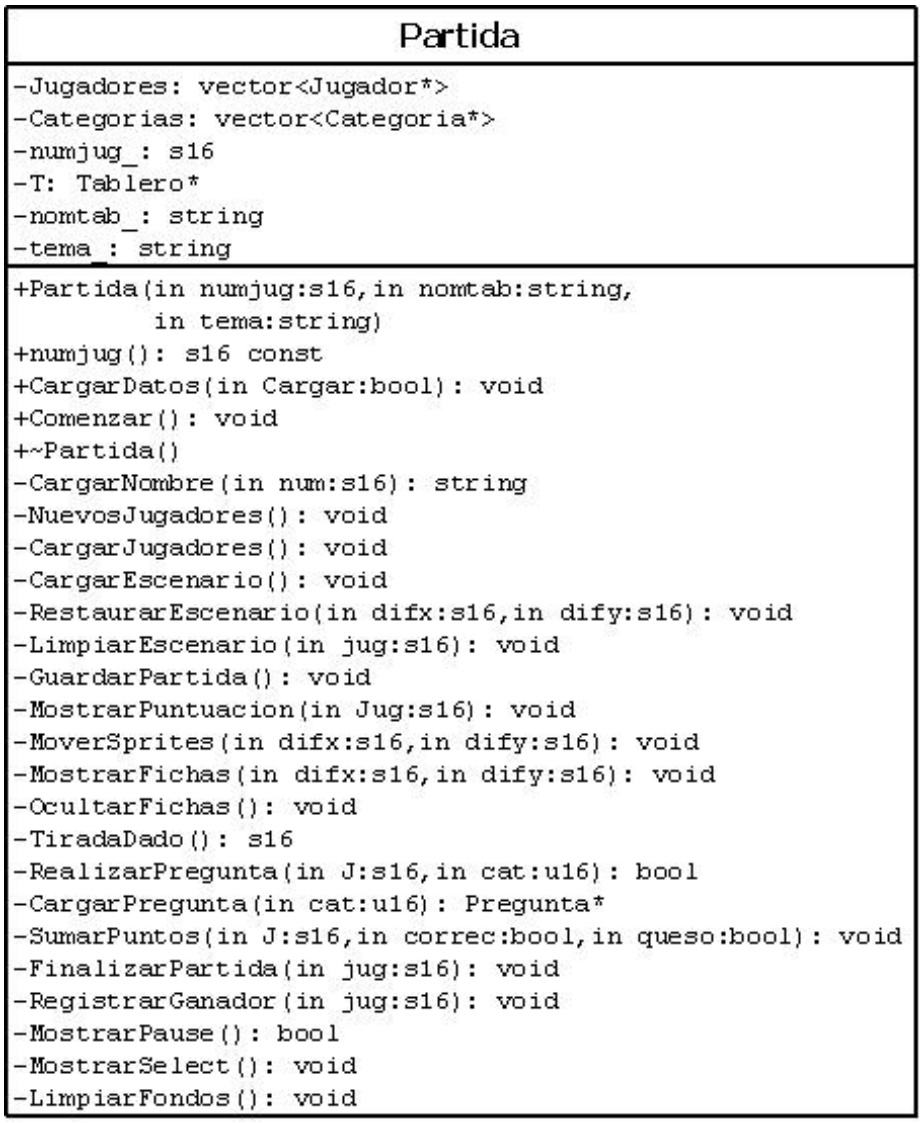


Figura 8.11: Clase Partida

Diseño e implementación

La clase "Partida" es la principal del juego y la más importante de todas. Representa a una partida creada del juego y se encarga de la gestión de toda la información que se recibe de los jugadores a través de los controles, la consulta, registro y eliminación de la información de la base de datos respecto a la partida y la gestión de todos los objetos que intervienen en una partida.

Constructor

En el constructor de la clase solamente se registra la información inicial que va a constar la partida, es decir, el nombre del tablero y el temario a usar, y el número de jugadores que van a participar en la partida.

Métodos

El método **CargarPartida** es la encargada de iniciar la partida y prepararla para su utilización. Para ello se crean los objetos de los jugadores a intervenir en la partida donde se puede realizar de dos maneras:

- La partida es nueva y por consiguiente se ejecuta el método **NuevosJugadores** que genera jugadores con valores iniciales, donde primero se solicita por pantalla, a través de un teclado, que se escriba un nombre para el jugador. Si no se escribe ningún nombre, el sistema otorga un nombre por defecto donde será llamado *Jugador X* donde X es el número del jugador en la partida.
- La partida es cargada desde la base de datos, donde se realiza la carga de datos de los jugadores desde la misma y se crean los objetos de los jugadores con los datos correspondientes, se ejecuta pues el método llamado **CargarJugadores**.

De las dos maneras se genera el vector de jugadores a utilizar durante la partida. A continuación se cargaran tanto el escenario como el temario que se usaran durante la partida. En la carga del escenario se carga tanto la imagen del tablero a usar y el objeto de Tablero que se cargara la información desde la base de datos, correspondiente al nombre del tablero recibido por el constructor, pone la pantalla en el centro del tablero y carga la pantalla de puntuación. En la carga del temario se cargaran todas las categorías con las preguntas correspondientes a ellas para guardarlas en un vector de objetos categorías. Por último si la partida no es cargada desde la base de datos todas las fichas de los jugadores se colocaran en la casilla central del tablero.

Durante el transcurso de la carga de la partida se mostrara en pantalla una imagen de carga y cuando finaliza la carga se carga y ejecuta la música que se escuchara durante el transcurso de la partida.

Una vez realizada la carga de la partida se procede a la ejecución del bucle principal del juego que se ejecutara hasta que finaliza la partida. El método encargado de ello se llama **Comenzar**. Para la explicación del bucle iremos por partes:

Inicialización Partida

bool fin = false, bucle = false, resp = false, salir = false;

Punto P(0, 0);

u16 dado = 0, jug = 0, num = 0, numcas = 0, cat = 0, cont = 0;

u32 anim = 0;

```
s32 difx = (T->ancho() / 2) - 128;
s32 dify = (T->alto() / 2) - 96;
```

Esta parte se especifica la inicialización de los valores de la partida al comenzar. Los valores booleanos *fin* y *salir* corresponden a la finalización de la partida donde *fin* es cuando una partida finaliza porque un jugador ha ganado la partida y *salir* es cuando un jugador en el menú de pausa finaliza la partida. El valor de *resp* indica si el jugador actual ha respondido o no correctamente la pregunta. Y el valor bucle indica si se ha finalizado el turno del jugador o no.

La variable *Jug* corresponde al jugador que actualmente tiene el turno.

Las variables *difx* y *dify* son dos variables que indican la distancia que se encuentra las coordenadas de la esquina superior izquierda de la pantalla inferior respecto a la esquina superior izquierda del tablero. Estas variables son usadas para el cálculo del movimiento del tablero y de los sprites sobre el mismo.

El resto de variables son variables temporales que se van modificando según transcurre la partida.

BUCLE PARTIDA

Inicialización Turno

```
if(cat != 7){
    OcultarFichas();
    MostrarJugador(jug);
}
MostrarFichas(difx, dify);
anim = 0;
resp = false;
bucle = false;
cont = 0;
dado = TiradaDado();
T->Sel_Casillas(Jugadores[jug]->casilla(), dado);
num = T->CrearLuces(difx, dify);
```

Cada vez que empieza un turno se inicializan ciertas variables para que el jugador que le toca pueda participar. Inicialmente se comprueba que el jugador no es el mismo, es decir que la casilla que anteriormente cayo el jugador que tenía el turno anterior, no cayó en una casilla de repetir tirada, entonces si el jugador no es el mismo del turno anterior se ocultan las fichas, con el método **OcultarFichas**, del tablero y se muestra una imagen con el jugador que le toca en el turno con el método **MostrarJugador**.

El método **MostrarJugador** Muestra una imagen con el jugador que le toca según el valor de la variable *jug* que recibe por parámetros y se queda a la espera a que el jugador correspondiente pulse en la pantalla para continuar la partida.

Una vez el jugador ha pulsado la pantalla para continuar se muestran de nuevo las fichas del tablero en la posición que tenían anteriormente con el método **MostrarFichas**, inicializa las variables temporales y se realiza una nueva tirada de los dados con el método **TiradaDado**.

El método **TiradaDado** crea en pantalla dos sprites que son la representación de los datos y se realiza una tirada automáticamente. Antes de realizar la animación de la tirada el sistema ejecuta dos veces un método aleatorio de la biblioteca PAlib[6] **PA_RandMinMax** que genera un número aleatorio dando un mínimo y un máximo. Una vez se sabe el número de los dos dados, se procede a la animación de los datos y la imagen final cambia dependiendo del valor que se haya obtenido. Una vez mostrado en pantalla el resultado, se eliminan los sprites de los datos y se devuelve el valor de la suma de los dos dados.

Una vez obtenido el valor de la suma de los datos se efectuara al cálculo de las posibles casillas que el jugador puede optar para mover su ficha. Para ello se ejecutan los métodos pertenecientes al tablero para determinar las casillas y se crean sprites de luces en las casillas que se devuelvan en la función. A las funciones del tablero se le pasa tanto la casilla actual del jugador, como el número de la suma de los dados.

Bucle Turno

Mover Pantalla

```
difx += (Pad.Held.Right - Pad.Held.Left) * 4;
dify += (Pad.Held.Down - Pad.Held.Up) * 4;
if(difx < 0) difx = 0;
if(difx > 256) difx = 256;
if(dify < 0) dify = 0;
if(dify > 192) dify = 192;
if((Pad.Held.Right - Pad.Held.Left) != 0 or (Pad.Held.Down - Pad.Held.Up) != 0)
    MoverSprites(difx, dify);
PA_EasyBgScrollXY(0, 3, difx, dify);
```

Cuando el jugador decide mover la pantalla a través del tablero, el sistema registra las pulsaciones de las flechas de dirección del pad de la consola y lo va guardando en las variables *difx* y *dify*, según las flechas de dirección que se hayan pulsado. Una vez se modifica las variables de posición de la pantalla si sobrepasa la pantalla el tablero ya sea horizontalmente o verticalmente se le otorga el valor máximo o mínimo a la distancia para que no sobresalga. Una vez determinada la posición de la pantalla respecto al tablero se mueven los sprites de toda la pantalla inferior para que conserven su posición sobre el tablero con el método **MoverSprites** que simplemente llama a las funciones de mover las fichas y las luces del tablero. Por último se mueve la imagen del tablero a la posición correspondiente.

Animar Luces

```
if(bucle == false){
    if(cont % 10 == 0){
        T->AnimarLuces(anim);
        anim++;
    }
    cont++; // Contador animación
}
```

Este fragmento del método se encarga de la animación de las luces cuando el juego está en espera a que el jugador seleccione una casilla a avanzar. Para que la animación no sea demasiado rápida, se avanza un paso en la misma cada diez frames de la consola, los frames están controlados con el contador *cont*. La consola tiene una velocidad de *60 fps*, por consiguiente, cada segundo se ven seis pasos de la animación.

Menú PAUSA

```
    if(Pad.Newpress.Start){
        LimpiarEscenario(jug);
        T->EliminarLuces();
        salir = MostrarPause();
        if (salir == false){
            T->CrearLuces(difx, dify);
            RestaurarEscenario(difx, dify);
            MostrarPuntuacion(jug);
        }
    }
```

Durante la partida, en los tiempos de espera de cada turno, se puede pulsar el botón *start* de la consola para acceder al menú de pausa del juego. Antes de ejecutar el menú, el juego limpia completamente de sprites y fondos las pantallas para la carga de las nuevas imágenes proporcionadas para el menú, este proceso se lleva a cabo por los métodos **LimpiarEscenario**, borra las imágenes de fondo, los textos y las fichas, y **EliminarLuces**, borra todos los sprites de las luces que marcan las casillas del tablero.

Una vez la pantalla este limpia se procede a Cargar el menú con el método **MostrarPause**, que realiza las siguientes opciones:

- Carga los fondos e imágenes del menú de pause en las pantallas.

- El menú tendrá tres opciones que el jugador podrá seleccionar o con el stylus o con las flechas de dirección. Las opciones son:
  - **Guardar Partida**: La partida se guarda en la base de datos sobrescribiendo lo que haya en la base de datos y da la orden al juego para salir de la partida.
  - **Continuar Partida**: Esta opción da la orden de salir del menú de pausa para continuar la partida por donde se quedo.
  - **Salir**: Esta opción da la orden al juego de salir de la partida sin guardar nada en la base de datos.

Una vez se sale del menú de Pausa según la opción elegida por el jugador habrá dos opciones:

- El jugador selecciono Guardar Partida o Salir: Donde la partida se cerrara volviendo al menú principal del juego
- El jugador selecciono Continuar Partida: Entonces el juego vuelve a cargar todas las imágenes del juego para continuar la partida. Esta carga está definida en los métodos
  - **CrearLuces**: Método del tablero que vuelve a colocar los sprites de las luces en las casillas seleccionadas en el tablero.
  - **RestaurarEscenario**: Vuelve a cargas las imágenes de fondo y las fichas del juego.
  - **MostrarPuntuacion**: Carga la información de la pantalla superior con los datos del jugador que es el turno.

menú SELECT

```
if(Pad.Newpress.Select){
    LimpiarEscenario(jug);
    T->EliminarLuces();
    MostrarSelect();
    RestaurarEscenario(difx, dify);
    T->CrearLuces(difx, dify);
    MostrarPuntuacion(jug);
}
```

Al igual que el menú de pausa, este menú se puede ejecutar en los tiempos de espera de cada turno del jugador. A diferencia del menú de pausa, esta opción muestra según los colores de las casillas del tablero, los nombres de las categorías de las mismas para que el jugador pueda consultar a que categoría corresponde la casilla que desea avanzar por si cambia de opinión. Antes de mostrar el menú de *Select*, el sistema limpia las pantallas para la carga de nuevas imágenes, eliminando tanto las imágenes como los sprites del mismo.

En la pantalla de *Select* aparecerán por orden las categorías del temario elegido por el jugador y junto a las categorías los colores de las casillas que corresponde la categoría. Para salir de esta pantalla se debe pulsar de nuevo la tecla *Select* de la consola.

Selección de casilla

```
for(u32 i = 0; i<num; i++){
    if (PA_SpriteTouched(30 + i)){
        numcas = T->Pulsar_Cas(i, P, cat);
        Jugadores[jug]->ficha()->MoverCasilla(P.x(), P.y(), difx, dify);
        Jugadores[jug]->casilla() = numcas;
        bucle = true;
        T->EliminarLuces();
    }
}
```

Una vez se ha lanzado el dado y se sabe a qué casillas se puede avanzar con la ficha desde la casilla que estamos actualmente, durante el tiempo de espera, el jugador debe elegir una de las casillas marcadas con luces por el sistema. El sistema por cada bucle del juego, correspondiente a un frame, está comprobando continuamente si el jugador pulsa una casilla, para esta comprobación se recorre el vector de sprites de luces que genera el tablero cuando se calcula las opciones, y gracias al método de PAlib[6] **PA_SpriteTouched** se puede comprobar si el jugador ha tocado el sprite con el stylus y que sprite ha sido. Una vez se sabe que sprite se ha pulsado, como hay relación directa entre el vector de sprites de luces y el vector de casillas seleccionadas por el sistema, se extrae el índice de la casilla pulsada, así como la categoría que pertenece y el punto central de la misma para su utilización con el método **T->Pulsar_Cas** y se llama a los métodos siguientes para el tratamiento de la información:

- **Jugadores[jug]->ficha()->MoverCasilla(P.x(), P.y(), difx, dify);**

  El sistema mueve la ficha del jugador que es el turno actualmente a la posición central de la casilla elegida. Se pasa también los parámetros de distancia de la pantalla por si la casilla elegida esta fuera de los límites de la pantalla.

- **Jugadores[jug]->casilla() = numcas;**

  El sistema actualiza la posición actual del jugador indicándole la casilla que se encuentra actualmente para su posterior utilización.

  Una vez se sabe la casilla elegida se da por finalizado el bucle del turno del jugador y se procede al análisis de la casilla que se ha elegido.

Análisis selección y carga de pregunta. Este es el último paso una vez se finaliza el turno de espera del jugador y se procede a formular la pregunta correspondiente a la categoría. Antes de avanzar se realiza una comprobación del tipo de casilla que ha elegido el jugador:

- **La casilla es la casilla inicial o de color blanco**: Estos dos tipos de casillas tiene la función de repetir la tirada del dado por parte del jugador para seguir avanzando por el tablero, es decir reinician el turno del jugador actual.
- **Las casillas de colores**: Continúan el turno formulando la pregunta de la categoría elegida por el jugador actual.

Si la casilla elegida es de color, es decir que esta asignada a una categoría del temario se procede a la carga de la pregunta perteneciente a la misma. Para ello se utiliza el siguiente fragmento de código:

```
LimpiarEscenario(jug);
resp = RealizarPregunta(jug, cat - 1);
if(resp == true){
    if (T->Es_Queso(numcas) and !Jugadores[jug]->queso()->Contiene(cat)){
        PA_LoadBackground(0, 3, &QuesoTexto);
        for(inti = 0; i< 200; i++)
            PA_WaitForVBL();
        PA_DeleteBg(0, 3);
        Jugadores[jug]->ficha()->AgregarQueso(cat);
        Jugadores[jug]->queso()->AgregarQueso(cat);
        if (Jugadores[jug]->queso()->Completo())
            fin = true;
    }
}else{
    if(numjug_ > 1){
        jug++;
        if(jug == numjug_)
            jug = 0;
    }
}
PA_ResetSpriteSys();
SumarPuntos(jug, resp, T->Es_Queso(numcas));
RestaurarEscenario(difx, dify);
```

Vamos a explicar paso a paso el proceso:

- **RealizarPregunta(jug, cat - 1):**
  - Este método es el encargado de todo el proceso de formulación de una pregunta. El primer paso es la selección, a través del objeto de la categoría seleccionada, de la pregunta a formular. Una vez seleccionada y cargada la pregunta, esta se muestra tanto en la pantalla superior, siendo el texto de la pregunta, como en la pantalla inferior, las posibles respuestas. Esta carga de la pregunta se realiza a través del método **CargarPregunta**, otra de las cosas que hace este método es cargar en la pantalla inferior una esfera de colores, simulando un reloj, que va disminuyendo los colores según pasa el tiempo.

- **Bucle Pregunta**: ya puesta en pantalla la pregunta formulada el juego se queda a la espera a que el jugador seleccione la respuesta correcta. Esta selección se deberá realizar con el stylus de la consola, donde el bucle comprobara si se realiza alguna pulsación, y según la zona de pulsación será la selección de una respuesta u otra. Si no se pulsa la pantalla el bucle tiene un contador que va aumentando cada frame, este contador si llega a sesenta se considera que ha pasado un segundo ya que la consola funciona a sesenta frames por segundo, cuando llega cinco veces el reloj disminuye un cuadro y así hasta doce veces, ya que el tiempo que dispone el jugador para responder es de un minuto. Si el jugador no responde cuando pase el minuto, se considerara que la respuesta es incorrecta, ya que no ha seleccionado ninguna respuesta.
- **Respuesta Pregunta**: Una vez seleccionada una respuesta o pase el tiempo, si la opción elegida es la respuesta correcta de la pregunta, se muestra por pantalla que se ha respondido correctamente y se devuelve al proceso principal que la pregunta se ha respondido correctamente o incorrectamente.

- El siguiente paso es el análisis del resultado donde pueden ocurrir dos opciones:

  - **La respuesta es correcta**: El sistema comprueba si la casilla, donde se respondió la pregunta, es una casilla principal además de comprobar si el jugador contiene o no la porción del queso del color de la casilla principal. Si es el caso entonces se notifica al jugador por pantalla que ha conseguido una porción del queso con el color de la categoría y se añade la porción a la ficha y al queso de la pantalla superior. El jugador conserva el turno y vuelve a tirar. Si el jugador ha completado las seis porciones del queso se da como finalizada la partida.
  - **Respuesta incorrecta**: El sistema calcula quien es el siguiente jugador quien le toca el siguiente turno.

- **SumarPuntos(jug, resp, T->Es_Queso(numcas));**

  - Tanto si se ha respondido o no correctamente la pregunta formulada, se reparte puntos al jugador actual, donde se otorgaran puntos negativos si se ha respondido incorrectamente, más exactamente diez puntos menos, y puntos positivos si se ha respondido correctamente, dando diez puntos si es una casilla normal y treinta puntos si es una casilla principal.

- Una vez terminado el proceso de la formulación de la pregunta se restaura el escenario principal del tablero y las fichas.

Finalizar Partida

```
if (salir == false){
    FinalizarPartida(jug);
    RegistrarGanador(jug);
}
```

Si uno de los jugadores ha completado las seis porciones del queso, quiere decir que ha ganado la partida. Una vez se sabe el ganador se da paso a la finalización de la partida realizando dos tareas:

- **FinalizarPartida(jug):** Muestra por pantalla el jugador que ha sido el ganador. Para ello carga en las dos pantallas los fondos correspondientes a la finalización de la partida y, para que sea más vistoso, el juego tiene implementado un sistema de fuegos artificiales. Este sistema genera cada segundo un par de objetos de fuegos artificiales, en una localización aleatoria en la parte inferior de la pantalla inferior. Una vez generado el fuego artificial, este se moverá verticalmente hasta la posición marcada aleatoriamente donde se procederá a ejecutar su animación de explosión. Una vez el fuego ha estallado se elimina de la pantalla. Si el punto marcado aleatoriamente para que estalle se encuentra en la pantalla superior, el fuego simplemente pasara de una pantalla a otra.
- **RegistrarGanador(jug):** Como el juego tiene un ranking con las mejores puntuaciones de los jugadores que han participado en las partidas, el ganador de una partida se registrara en la base de datos con su puntuación, para que se pueda mostrar posteriormente en la sección *ranking* del menú principal. Este registro solo se llevara a cabo si el jugador se encuentra entre los 10 mejores de todos los registros que contiene la base de datos de partidas anteriores.

8.3. Implementación menú principal del juego

8.3.1. Introducción y menú

Al inicial el juego desde la consola son los métodos que el sistema ejecuta automáticamente para dar comienzo al juego:

- **voidPresentacion(void):** Muestra por pantalla una pequeña introducción, a modo de simulación de empresa de videojuegos, de quien ha sido el creador del juego.
- **voidIntroStart(void):** Método encargado de la carga de los fondos y la música que va a tener el juego durante el menú del mismo. El fondo de nubes que se carga se inicializa para que tenga un movimiento lateral constante.
- **voidMenu_func(void):** Método que muestra por pantalla el menú principal del juego. Durante la muestra del menú aun se conserva el movimiento lateral del fondo de nubes y se entra en un bucle a la espera de que el jugador seleccione una opción. Dicha selección se comprobara desde los métodos **Stylus.Newpress** y **Pad.Newpress.A**, que darán paso a la ejecución de la opción elegida ya sea desde el stylus o desde el pad de control de la consola. El jugador también puede optar por la selección de la opción a través del pad de control moviendo un sprite en forma de rombo por las opciones del menú.
- **voidRestaurarMenu(void):** Método usado para la restauración del menú una vez que se finaliza una partida.

8.3.2. Opciones del menú

Según la opción elegida por el jugador en el menú principal del juego, se pueden ejecutar las siguientes opciones:

- **Nueva Partida:** Esta opción contiene ciertos apartados que se llevan a cabo para el inicio de una nueva partida en el juego:

  - **Selección número de jugadores ("NumJugadores()")**:El sistema carga las imágenes y opciones para que el jugador seleccione la cantidad de jugadores que jugará la partida. Esta selección se realiza o bien con el pad de control o con el stylus y se pueden elegir hasta cuatro jugadores.
  - **Selección de tablero ("ElegirTab()")**: Se carga desde la base de datos los posibles tableros que se pueden elegir para jugar en el. Una vez se sabe los tableros se muestran en pantalla para que el jugador pueda elegir entre los posibles y continuar con la configuración de la partida. Esta selección se bebe hacer pulsando con el stylus los botones y pulsando el botón *A* para continuar.
  - **Selección de Temario ("ElegirTema()")**:Es el mismo menú para la selección de tableros pero esta vez se cargan desde la base de datos, los temarios disponibles para su utilización en la partida.
  - **Crear Partida**: Una vez que se ha configurado la partida, se crea el objeto partida con la información elegida y se da paso a la ejecución del juego.

- **Cargar Partida:** La opción cargar partida se utiliza cuando hay una partida guardada en la base de datos para su continuación. Como solo puede haber una partida a la vez guardada, una vez se ejecuta la opción se procede a la carga de la partida desde la base de datos en el objeto de la clase Partida, para posteriormente ejecutar la partida y continuar por donde se quedo. Todo esto se lleva a cabo desde el método Cargar partida de la clase Partida.

- **Mostrar Ranking:** Esta opción carga desde la base de datos todos los ganadores registrados en la tabla de ranking para mostrarlos por pantalla. Para ello carga los fondos correspondientes y coloca en columnas los datos de los jugadores en orden descendentes de puntos. Una vez que el jugador pulse el botón Start o pulse con el stylus, se volverá al menú principal.

- **Mostrar Créditos:** Esta opción es añadida por el creador del juego para mostrar al jugador información de quien ha sido el desarrollador y ayudas que ha recibido.

Capítulo 9

Pruebas

Aunque se haya especificado las pruebas por cada clase y que se ha hecho en ellas, el juego ha sido probado en su totalidad por testeadores que han comprobado el funcionamiento completo del juego. Estas personas tienen los perfiles siguientes:

- **Sujeto 1:** Mujer 27 años, estudios de ciclo formativo de grado superior de secretariado, nivel de informática usuario alto, con conocimiento de ofimática, con experiencia en juegos de mesa y de ordenador y conocimientos en varios temas de cultura.
- **Sujeto 2**: Mujer 17 años, estudios de ciclo formativo de grado medio de comercio, nivel de informática usuario medio, con experiencia en juegos de mesa y de ordenador y pocos conocimientos en temas de cultura.
- **Sujeto 3**: Hombre 48 años, estudios de formación profesional de electrónica, nivel de informática usuario medio, poca experiencia en juegos de ordenador y algo de juegos de mesa y conocimientos de temas de cultura.
- **Sujeto 4**: Hombre 23 años, estudiando curso universitario de diplomatura de electricidad, nivel de informática usuario alto, mucha experiencia en juegos de ordenador y de mesa y conocimientos en temas de cultura.

Se les ha propuesto que realicen las siguientes opciones, ya sea individualmente o conjuntamente:

- Creación de una partida de un solo jugador: Con la finalidad de comprobar que el juego funciona completamente con un jugador y que se finaliza cuando el jugador consiga todas las porciones.
- Creación de una partida de varios jugadores: Con la finalidad de comprobar que el juego gestiona bien los turnos y los puntos de cada jugador, así como las pociones que se han conseguido según pasan los turnos. Y finalmente se comprueba que el juego finaliza y se registra el ganador una vez finaliza la partida.
- Comprobación del funcionamiento de las distintas opciones del juego como son el ranking y los créditos, pero la más importante es la carga de una partida desde la base de datos que se haya guardado anteriormente.

Todos los sujetos consiguieron realizar con éxito todas las opciones propuestas, donde, con la ayuda de los sujetos que probaron el juego, se pudo mejorar la manejabilidad del juego, ya que en algunas zonas era complejo intuir que botones había que pulsar para continuar con la partida. Aparte de la mejora de la manejabilidad, también ayudo su opinión sobre los colores, tanto del menú como del juego, para que no sea molesto para la vista, y tenga el juego mejor visibilidad.

Capítulo 10

Conclusiones

10.1. Aspectos generales

El juego que se obtiene como resultado del proyecto, es una simulación de lo que sería un juego de mesa basado en un juego de preguntas y respuestas en una de las consolas más vendidas hasta el momento como es la Nintendo DS. Este juego estará a disposición tanto como el código como el juego en internet libre para que la gente que desee jugar al juego, como aprender el uso de las bibliotecas de programación de la Nintendo ds puedan hacerlo con total libertad.

Con el proyecto quiero dar a entender, sobre todo a la gran comunidad de jugadores, que un juego para una consola no tiene por qué ser única y exclusivamente hecho por empresas grandes ni multinacionales, sino que cualquier usuario con los conocimientos necesarios de programación puede realizar su propio juego para su disfrute o el disfrute de amigos y compañeros.

10.2. Conocimientos adquiridos

Además del refinamiento a la hora de la programación en un lenguaje tan completo como es el lenguaje C++[5], he conseguido obtener un gran conocimiento a la hora de la programación orientada a objetos y su estructuración a la hora de organizar las clases y sus relaciones.

Otro aspecto a considerar es la adquisición de conocimiento de las diversas bibliotecas repartidas en internet para la programación en consolas, más concretamente de la Nintendo DS, ya que tanto la biblioteca PAlib[6] como libNDS[2] han resultado ser unas bibliotecas completas y útiles a la hora de gestionar todos los aspectos de hardware de la consola para hacer posible el juego desarrollado. Para ello se ha tenido que investigar múltiples paginas y tutoriales que explicaban más o menos el funcionamiento de estas bibliotecas y como aplicar los métodos correspondientes para el transcurso del juego. También se ha adquirido conocimientos para la creación de un archivo Makefile para la compilación en Linux del proyecto para generar el juego, donde cierto es que ya venía definido con el proyecto por defecto en un principio, pero la configuración del mismo para la inclusión de diversas bibliotecas adicionales y otros aspectos ha sido muy útil.

Respecto a la base de datos del juego, se ha adquirido conocimiento de la creación y gestión de una base de datos Sqlite3[11] para realizar todas las gestiones de la información del juego a través de consultas SQL. Junto a la creación de la base de datos y la inclusión al proyecto se ha aprendido la forma de crear objetos de clase "singleton" para que solo haya un único objeto compartido en el sistema de una clase. Y por último el manejo de sentencias SQL ha sido conseguido para una mejor gestión de la base de datos tanto para el juego como demás gestiones que se realizan en el proyecto.

Por último, externamente al juego, se ha adquirido conocimientos diversos a la hora de la creación de herramientas para el proyecto.

- Conocimientos para la creación de un programa de gestión basado en visual basic .NET para la gestión de la base de datos del juego, así como la creación de la información que compone un tablero.
- Conocimientos de Latex para la creación de la documentación del proyecto, usado junto al programa Scientific Workplace, para generar documentos pdf.
- Uso del Photoshop y Gimp para el dibujo de los diferentes fondos y sprites que componen el juego.
- Uso del Doxygen[3] para generar la documentación respecto a los comentarios del programa para colocarlos tanto en la web como en el programa.
- Y por último el manejo del programa Dia para la creación de diagramas tanto de la base de datos como los diagramas de clases.

10.3. Posibles mejoras

Las posibles mejoras para el proyecto sería el siguiente:

- **Añadir Tableros Personalizados**: Aunque el juego admita múltiples tableros desde la base de datos, con la biblioteca actual que se usa para la carga de imágenes no es posible que un usuario pueda añadir un tablero personalizado. Por ello uno de las mejoras sería conseguir un programa de gestión que agregue tanto a la base de datos como al juego de tableros que el usuario desee jugar.
- **Mejorar Programa de gestión**: Aunque se disponga de un programa de gestión de la base de datos, esta gestión es solamente de las categorías del juego, ya que es lo único que actualmente es útil modificar. En un futuro se podría realizar un programa más completo del juego, ya sea en visual basic .NET para Windows o en C++[5] con el borland C++[5] para poder ejecutarlo además en Linux.
- **Mejorar imágenes**: Las imágenes del juego están hechas por un aficionado del diseño de las imágenes, y se nota que son de baja calidad, una posible mejora seria la incorporación de imágenes hechas por alguien experto en la materia para que el juego se mostrara mas profesionalmente.

10.4. Futuro del proyecto

Aunque las consolas se quedan obsoletas según pasan los años, el diseño del juego se podría extrapolar a otros sistemas para su adaptación y uso. Otro futuro del proyecto seria adaptar las funcionalidades del juego a la misma consola pero de las más avanzadas, ya que las Nintendo DS antiguas cada vez serian menos, y así disponer de más recursos para añadir nuevas funcionalidades al juego.

El juego también se podría coger como referencia para futuros proyectos que se requiera de una base de datos en programación orientada a objetos en C++[5] por otros usuarios que deseen crear su propio juego.

Bibliografía

[1] Página principal de DevkitPro.
`http://devkitpro.org/`

[2] Sección para la programación de la NDS de DevKitPro.
`http://devkitpro.org/wiki/Getting_Started/devkitARM`

[3] Página principal de doxygen.
`http://www.doxygen.org`

[4] Lambert M. Surhone, Mariam T. Tennoe y Susan F. Henssonow.
Doxygen: Documentation Generator, C+ +, Java(programming language), Objective- C.
ISBN-10: 3639910028

[5] G. Aburruzaga García, I. Medina Bulo, F. Palomo Lozano.
Fundamentos de C++.
Servicio de Publicaciones de la Universidad de Cádiz, 2001.

[6] Página oficial de la libreria PAlib, dada de baja por el propietario.
`http://www.PAlib.com`

[7] Página de la librería PAlib en devkitpro.
`http://devkitpro.org/wiki/PAlib`

[8] Forja de la librería de PAlib.
`http://sourceforge.net/projects/pands/`

[9] Página del proyecto Quiz Libre.
`http://quizlibre.forja.rediris.es/`

[10] Forja del proyecto Quiz Libre en RedIris.
`https://forja.rediris.es/projects/quizlibre/`

[11] Página oficial de Sqlite.
`http://www.sqlite.org/`

[12] SQLite.
Chris Newman.
ISBN-10: 067232685X.

[13] Página de música libre en formato MOD para juegos.
http://modarchive.org/

[14] Artículos de wikipedia sobre videoconsolas y juegos de mesa.
http://es.wikipedia.org/wiki/Juegos_de_mesa
http://es.wikipedia.org/wiki/Videoconsolas

Apéndice A

Software Utilizado

Adobe Photoshop CS4 Photoshop es un programa de diseño y modificación de imágenes de Adobe, donde se pueden editar cualquier tipo de imagen para usarlas en el proyecto.

Adobe DreamWeaver CS4 DreamWeaver es un framework de programación en HTML y CSS para el diseño de páginas web.

GIMP Gimp es un programa de diseño y modificación de imágenes. El programa es software libre distribuido por internet para procesar imágenes de cualquier formato.

Dropbox Dropbox es un software utilizado para el almacenamiento de datos en un disco duro virtual en internet. Es usado para tener cualquier archivo disponible desde cualquier ordenador.

Tortoise SVN TartoiseSVN es un software de control de versiones gratuito que permite gestionar los cambios realizados en el proyecto. Estos cambios pueden ser realizados por uno o varios desarrolladores.

PAGfx PAGfx es un programa proporcionado por los creadores de PAlib[6] para el procesamiento de imágenes a código C, para su utilización mediante métodos de la biblioteca PAlib[6].

Total audio converter Total Audio Converter es un programa rápido y eficaz para el tratamiento de audio y música para poder incorporarlo al juego. El programa no es libre pero proporciona una versión mínima que se puede trabajar con el.

Microsoft visual c++ 2008 Microsoft visual c++ 2008 es un framework de Microsoft para el desarrollo de aplicaciones en lenguaje C++[5].

Dia Dia es un editor de gráficos vectoriales el cual incluye distintas plantillas para distintos tipos de gráficos, como pueden ser UML, ERe, diagramas de flujo, esquemas Cisco de red y un larguísimo etcétera.

DevkitPro DevKitPro[1] es un conjunto de compiladores, bibliotecas estándar de C y C++[5] y varias utilidades que permiten compilar código fuente escrito en esos lenguajes para poder ser ejecutado en varias videoconsolas actuales.

Sqlite3 SQlite[11] es biblioteca libre para la creación de una base de datos y gestionarla con sentencias SQL desde un programa en varios lenguajes de programación, entre ellos C++[5].

PAlib PAlib[6] es una biblioteca de alto nivel, escrita en C, que gestiona todos los aspectos del manejo del hardware de la consola Nintendo DS. Esta realizada por varios usuarios en internet y trabaja junto a libnds[2] de devkitpro[1].

Scientific Workplace Scientific Workplace es un programa para la edición de documentos escritos en Latex para una mayor facilidad a la hora de añadir formato al documento.

GNU Make GNU Make es el programa de recompilación y de control de dependencias por excelencia. Se puede utilizar para compilar proyectos software en diversos códigos.

Doxygen Doxygen[3] realiza una documentación automática de código fuente. Puede generar esta documentación en varios formatos, incluyendo HTML y Latex.

Apéndice B

Manual de la biblioteca PAlib

B.1. Instalación en Windows

B.1.1. Configuración del Entorno de Desarrollo y Emuladores

Antes de comenzar a escribir programas con la ayuda de esta librería es necesario tener instalado y configurado una serie de herramientas.

La primera utilidad a instalar es devkitpro[1] que se define como un conjunto de herramientas para desarrollar videojuegos homebrew. Actualmente da cobertura a las siguientes plataformas: PlayStation Portable (PSP), Game Cube, Game Boy Advance, GP32, y por supuesto Nintendo DS. Una vez descargado el fichero de instalación ejecutable desde su página web, se instala teniendo en cuenta que tan sólo es necesario las librerías que hacen referencia a Nintendo DS siendo estas las que están bajo el menú devkitARM. En el momento de escribir esto, la última versión disponible de devkitpro[1] es la 1.4.4 la cual incorpora el devkitARM release 20.

Para poder usar aplicaciones incluidas con PAlib[6] como son PAGfx (conversor de gráficos), PAFS (sistema de ficheros) o VHam (Ide de desarrollo); es necesario tener instalado .Net Framework.

PAlib[6] lo puedes encontrar por internet ya que la página oficial ya no existe, así que se tiene que buscar entre los círculos de programadores aficionados a la programación de consolas. Se recomienda instalarla en la misma carpeta que devkitpro[1] para que todo funcione correctamente. También que se instalen devkitpro[1] y PAlib[6] en rutas que no contengan espacios. Finalmente reiniciar el ordenador.

Para comprobar que se ha instalado correctamente, vamos a intentar compilar un código de ejemplo. Dentro de PAlibExamples\Text\Normal\HelloWorld arrancar el archivo por lotes build.bat si todo ha ido correcto se generan al menos tres archivos: HelloWorld.nds, HelloWorld.sc.nds y HelloWorld.ds.gba. Si ha sucedido algún error lo más probable es que haya que configurar las rutas. Considerando que el error es parecido a este: *make main.c arm-eabi-gcc.exe: CreatProcess: No such file or directory make[1]: * [main.o] Error 1 make: * [build] Error 2,* entonces hay que indicar al sistema operativo dentro del path donde se encuentran los binarios que hacen falta para realizar la compilación. Para ello hay que acceder a Propiedades Avanzadas de Mi PC -> Variables de entorno ->Panel de Control y en las

variables del sistema, modificar la variable Path (o crearla si no existe) para que incluya los siguientes directorios:

C: \devkitPro\ devkitARM\ bin;

C:\ devkitPro\devkitARM \arm-eabi \bin;

C:\ devkitPro \devkitARM \libexec\gcc \arm-eabi \4.1.1.

Suponiendo que devkitpro[1] está instalado en el directorio raíz *C:\devkitPro*.

Si ya se han conseguido generar los ficheros .nds, .sc.nds y .ds.gba es hora de arrancarlos. La opción más cómoda para el desarrollador es usar un emulador. Existen varios funcionales, entre los mejores está No$GBA que tiene una versión comercial y otra gratuita pero con restricciones, además de otros como Dualis, iDeas, DSemu.

Algunos vienen incluidos con la instalación de DevkitPro[1]. Hay que decir que ningún emulador funciona al 100 %. Sin embargo, se ha comprobado que en algunos emuladores funcionan mejor unas características, mientras que otras no tienen un buen resultado, por tanto lo mejor es tener instalado al menos dos emuladores distintos y por supuesto no fiarse de los resultados que estos puedan ofrecer, teniendo que testear las aplicaciones en el hardware de la videoconsola NDS cuando sea posible para cerciorarse de que funcione correctamente.

Otra parte importante es la edición del código. Existen distintos editores con los que se puede programar para DS. En principio bastaría con un editor de texto simple como el bloc de notas y luego utilizar alguna herramienta de generación de código como Make que relacione los fuentes con el compilador, pero es preferible usar un editor inteligente que tenga utilidades como coloreo de código, sugerencia de nombres reservados, vinculación con Make, que pueda ejecutarse el código directamente desde el entorno sobre un emulador, o incluso que con la ayuda de un script se envíe el compilado a la DS... Junto con devkitpro[1] se incluyen varios editores que ofrecen algunas de estas posibilidades. Quizás el más simple sea Programmers Notepad 2 que colorea el código según los estándares de C/C++[5] y es posible compilar sin salir del entorno. El otro editor incluido es Visual HAM que además de colorear código y compilar, da la posibilidad de ejecutarlo en alguno de los emuladores que estén instalados y configurados con el entorno. Otra opción más potente es usar Microsoft Visual Studio C++[5] ya sea en su versión completa o versión express. Este editor no viene incluido con devkitpro[1] y habrá que obtenerlo aparte. La versión express, Microsoft Visual C++ Express Edition, es gratuita y se puede descargar desde la página de Microsoft. Una vez configurado correctamente aparecerá un nuevo tipo de proyecto NintendoDS, Palib Application:

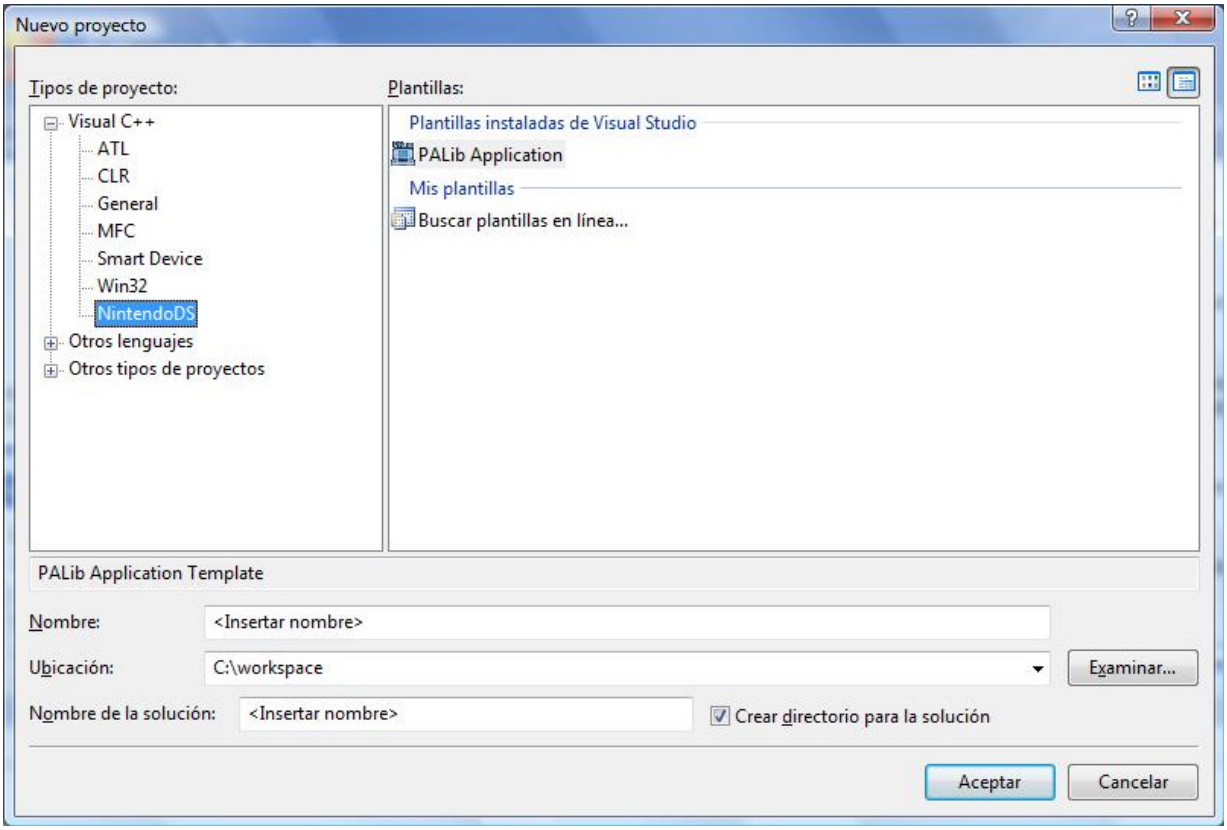


Figura B.1: PAlib configurado en Visual Studio. Fuente: Palib.info

Ahora el programador tendrá todas las facilidades de este editor como por ejemplo hacer uso de la herramienta IntelliSense, o crear un script para que el compilado se envíe al dispositivo de almacenamiento de DS.

Existe alguna utilidad interesante como por ejemplo instalar un servidor ftp en la videoconsola al que poder conectarse con un cliente ftp vía wifi para enviar los ficheros compilados sin tener que sacar de la DS el cartucho y la tarjeta de memoria. Esta cómoda utilidad se llama DSFTP y en su página web se puede encontrar una guía para hacerla funcionar.

Lo siguiente es desarrollar aplicaciones que abarquen los aspectos más importantes de la librería para aprender a utilizarla.

B.2. Instalación en Linux

Para instalar DevKitPro[1] en Linux, primero deberemos instalar todos los archivos necesarios para el sistema:

- devkitArm - El compilador.

- libnds - La librería principal.
- libfat-nds - Librería para manejar ficheros.
- dswifi - Librería para las funciones de red.
- maxmod - Librería de sonido.
- libfilesystem
- Default arm7
- Ejemplos NDS

Una vez descargados, creamos la carpeta donde se alojará y descomprimimos el compilador, por ejemplo en /opt.

```
sudo mkdir -p /opt/devkitpro
sudo chmod 777 /opt/devkitpro
cd /opt/devkitpro
tar -xvjf [devkitARM].tar.bz2
```

Tras ello, creamos una carpeta llamada libnds[2] en la ruta del devkitpro[1] y descomprimimos en ella todas las librerías y los datos del ARM7:

```
mkdir libnds
cd libnds
tar -xvjf [libnds].tar.bz2
tar -xvjf [libfat-nds].tar.bz2
tar -xvjf [dswifi].tar.bz2
tar -xvjf [maxmod].tar.bz2
tar -xvjf [libfilesystem].tar.bz2
tar -xvjf [default arm7].tar.bz2
```

Para los ejemplos, creamos una carpeta propia y los descomprimimos en ella:

```
cd ..
mkdir -p examples/nds
cd examples/nds
tar -xvjf [ejemplos nds].tar.bz2
```

El árbol de carpetas resultante se parecerá al que sigue (puede variar entre versiones):

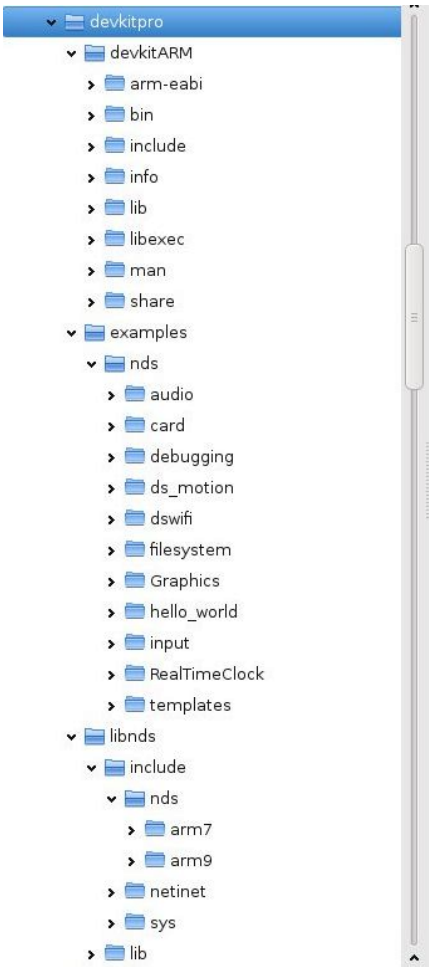


Figura B.2: Sistema de carpetas en linux. Fuente: Palib.info

Por último, hay que añadir las variables al entorno, editando el .bashrc, añadiendo al final las siguientes líneas:

```
export DEVKITPRO=/opt/devkitpro
export DEVKITARM=$DEVKITPRO/devkitARM
```

Para comprobar que la instalación se ha realizado correctamente, basta con ir a $DEVKITPRO/examples/nds/hello_world/ y ejecutar la orden make. Si todo va bien, tendremos un hello_world.nds listo para probar en emulador o en consola con flashcart.

Nota: se recomienda quitar al grupo Otros el permiso de escribir una vez acabada la instalación.

Nota: Las versiones de las librerías para este documento son:

- libnds: 1.5.0
- libfat-nds: 1.0.9
- dswifi: 0.3.13
- maxmod: 1.0.6
- libfilesystem: 0.9.9
- Default arm7: 0.5.20

B.3. Instalación de PAlib

La instalación de PAlib[6] es idéntica en sistemas Linux y Windows, ya que solo requiere bajarse la última versión (en nuestro caso PAlib 100707) y descomprir el contenido del paquete en /devkitPro/PAlib/.

El paquete PAlib[6] trae además una colección de herramientas útiles. A continuación comentaré PAGfx, herramienta pensada para convertir imágenes a un formato adecuado para usar con PAlib[6].

B.4. Programación en Nintendo DS usando PAlib

B.4.1. Plantilla PAlib

Palib[6] incluye una carpeta llamada PAlibTemplate que lleva dentro la estructura más básica pero completamente funcional de una aplicación para DS. Esta plantilla se compone de los siguientes ficheros:

- **Makefile**: Fichero que compila el proyecto. No es necesario modificarlo aunque tiene variables internas por si hace falta.
- **clean.bat**: Limpia el directorio de los archivos generados por posibles compilaciones anteriores.
- **make.bat**: Primero realiza la acción clean y luego hace una compilación del proyecto.
- **logo.bmp y logo_wifi.bmp**: Se trata de dos imágenes que representarán los iconos de la aplicación cuando se vaya a transferir el archivo compilado vía wifi. El primero debe tener 32 x 32 pixeles de tamaño y una paleta de diesiséis colores, mientras que el segundo es de 10 x 16 colores y sólo puede tener dos colores: blanco y negro.
- **Template.pnproj**: Es el fichero de proyecto para el editor Programmers Notepad 2, si se abriera este fichero con dicho programa, se abriría con la estructura de ficheros del proyecto.
- **Project.vhw**: Igual que Template.pnproj pero para el editor Virtual Ham.
- **Carpeta source**: Contiene el fichero main.c que será ejecutado como el inicio de la aplicación. En él viene un código que carga e inicializa la librería y que comentaremos más adelante.

- **Carpeta include**: Lugar donde se sitúan los ficheros de cabecera .h de C++[5].
- **Carpeta data**: En esta carpeta se incluyen los recursos necesarios por la aplicación, como pueden ser imágenes, archivos de audio, modelos 3D o ficheros binarios en general.

B.4.2. Primera aplicación

Una vez vista la plantilla, comenzaremos explicando su código para incluir más adelante alguna cosa más y terminar compilándolo y ejecutándolo. Este es el código de main.c:

```
// Includes
#include <PA9.h> // Include para PA_Lib
// Función principal
int main(int argc, char ** argv){
    PA_Init(); // Inicializa PA_Lib
    PA_InitVBL(); // Inicializa un est\U{e1}ndard VBL
    // Bucle infinito para mantener el programa en ejecución
    while (1){
        PA_WaitForVBL();
    }
    return 0;
} // Fin main()
```

La primera línea es una directiva include y aparecerá siempre que vayamos a usar la librería PAlib[6], PA9.h. Ya dentro del main llama a dos funciones de inicialización de la librería (se distinguen porque comienzan por PA_). La primera, **PA_Init()** es una inicialización general y si no se pusiera no funcionaría correctamente PAlib[6]. La siguiente línea es **PA_InitVBL()** que hace referencia al VBL (Vertical Blank Line) que es el tiempo que tarda la pantalla en refrescarse. Al inicializarlo permite al programa sincronizarlo con la pantalla, a *60 fps* (frames por segundo). Si no se hiciera esto la aplicación iría más rápido pudiendo incluso dejar de funcionar. Por tanto, estas dos líneas serán siempre fundamentales junto con el primer include.

Lo siguiente que aparece en el código es un bucle infinito que se corresponde con el ciclo de ejecución de la aplicación. Hasta este punto la aplicación debe haberse encargado de inicializar todos los datos y objetos necesarios, para que después se pueda acceder a este bucle, que se encargará básicamente de analizar los eventos que sucedan para actualizar la información que se tenga que mostrar por la pantalla. En este caso dentro del bucle sólo existe una línea que hace referencia a algo comentado anteriormente. Se trata de **PA_WaitForVBL()**, esta es la línea que hace sincronizar la aplicación a *60 fps*, situándola al final del bucle se asegura que este se repetirá con la periodicidad indicada. Por último se realiza una llamada a *return 0* para acabar correctamente con la aplicación, aunque recordemos que previamente existe un bucle infinito por tanto nunca se llegará hasta aquí a no ser que se salga explícitamente del bucle, con un break por ejemplo, lo cual no es nada deseable.

Si ejecutáramos esta aplicación no aparecería nada, sólo una pantalla en negro. Esto es correcto puesto que no se ha hecho uso de ninguna función que imprima por pantalla. Esta será la próxima tarea.

B.4.3. Textos

Texto Simple

En la documentación de la librería existe un módulo que hace referencia a mostrar textos por pantalla, este es Text Output System. Aquí aparecen todas las funciones necesarias para esta tarea. Un ejemplo de uso se puede ver en el siguiente código:

```
// Includes
#include <PA9.h> // Include para PA_Lib
// Función: main()
int main(int argc, char ** argv)
{
    PA_Init(); // Inicializa PA_Lib
    PA_InitVBL(); // Inicializa una estándard VBL
    PA_InitText(1, 2);
    PA_OutputSimpleText(1, 1, 2, "Hola Mundo DS");
    // Bucle infinito para mantener el programa en ejecución
    while (1){
        PA_WaitForVBL();
    }
    return 0;
} // Fin main()
```

He aquí una aplicación que inicia la librería correctamente, para poder mostrar un texto también hace falta iniciar dicho módulo con **PA_InitText**. Después hay que llamar a la función **PA_OutputSimpleText** para que se muestre el texto por la pantalla. Sin embargo, para usar estas funciones hace falta indicarles una serie de parámetros. La primera función, **PA_InitText (u8 screen, u8 bg_select)** tiene dos parámetros. El primero es para indicar en cual pantalla se va a pintar *(0 = inferior, 1 = superior)*, y el segundo hace referencia al fondo dentro de la pantalla donde se pintará *(0 - 3)*. De momento basta con saber que NDS puede mostrar hasta cuatro fondos por cada pantalla, donde cada fondo tiene su propia paleta de doscientos cincuenta y seis colores. Estos fondos pueden ser formados por tiles o bien de ocho o dieciséis bits de información. La otra función es **PA_OutputSimpleText (u8 screen, u16 x, u16 y, const char *text)**. El primer parámetro representa lo mismo, la pantalla. Los dos siguientes son las coordenadas dentro de la pantalla, X es la posición horizontal e Y es la vertical. La posición (0, 0) se corresponde con la esquina superior izquierda y los valores positivos van hacia la derecha para la X y hacia abajo para la Y. La esquina inferior derecha tiene como coordenada

(255, 191), que se corresponden con las dimensiones de la pantalla en píxeles, *256 x 192*. Sin embargo, para hacer referencia a las posiciones del texto debemos darnos cuenta de que el texto son imágenes cargadas en tiles, por tanto si un tile está formado por *8 x 8 píxeles*, entonces la pantalla está formada por *256/8 x 192/8 = 32 x 24 tiles* siendo *(31, 23)* el último tile donde se podrá escribir texto en la pantalla. El último parámetro es el texto que se quiere mostrar. El código anterior se corresponde con la aplicación que muestra el texto "Hola Mundo DS".

De este modo se puede imprimir texto de una manera simple, pero existe otra función muy importante que hace las veces de printf, esta es **PA_OutputText (u8 screen, u16 x, u16 y, char *text, arguments...)**. Los primeros cuatro parámetros son iguales, pero además, al final se pueden incluir una serie de argumentos que irán incluidos dentro de la cadena text. Dependiendo del tipo que se vaya a mostrar se usarán unos indicadores u otros:

- **Integer**: %d, Ejemplo: PA_OutputText(1, 0, 0, "valor: %d", 3);
- **Float**: PA_OutputText(1, 0, 0, "Float value : %f3", 4.678);
- **String**: PA_OutputText(1, 0, 0, "Hola %s", "mundo");

Del mismo modo que en los argumentos se pasan valores constantes, se podrían pasar variables que contengan valores de cada tipo.

Box Text

El texto simple puede ser útil, pero a menudo, no se corta en unos límites que desearíamos. A veces, se necesita envolver el texto, confinado a un área rectangular (box) de la pantalla. Para ello, está **PA_BoxText (u8 screen, u16 basex, u16 basey, u16 maxx, u16 maxy, const char *text, s32 limit)**.

Sus parámetros se explican a continuación:

- **screen** - la pantalla a usar *(0 ó 1)*
- **basex** - X de la esquina superior izquierda del área, en tiles
- **basey** - Y de la esquina superior izquierda del área, en tiles
- **maxx** - X de la esquina inferior derecha del área, en tiles
- **maxy** - Y de la esquina inferior derecha del área, en tiles
- **text** - texto a visualizar
- **limit** - máximo de letras a mostrar en este tiempo (puede usarse también para escritura lenta del texto)

Esta función devuelve el número de letras realmente sacadas.

Para ralentizar la escritura, aumenta el límite en un punto. Cuando el número devuelto no cambia, ha parado (porque no hay más texto ó bien porque se ha alcanzado el límite). En este punto, puedes dejar de aumentar el límite.

Color del Texto

Hay tres formas de elegir el color del texto.

PA_SetTextCol En cualquier momento, puedes cambiar el color del texto (fuente) usando la función: **PA_SetTextCol(screen, r, g, b)**. Esta establece el color que necesites en una pantalla. Todo el texto en esa pantalla tendrá el mismo color.

Aquí tienes algunos ejemplos:

- **Azul**: PA_SetTextCol(screen, 0, 0, 31);
- **Rojo**: PA_SetTextCol(screen, 31, 0, 0);
- **Blanco**: PA_SetTextCol(screen, 31, 31, 31);
- **Negro**: PA_SetTextCol(screen, 0, 0, 0);
- **Gris**: PA_SetTextCol(screen, 22, 22, 22);
- **Magenta**: PA_SetTextCol(screen, 31, 0, 31);

PA_SetTextTileCol **PA_SetTextTileCol** es ligeramente diferente a **PA_SetTextCol**, ya que esta no cambia el color de los textos ya existentes, permitiéndote de esta forma, tener diferentes colores en la misma pantalla.

Un ejemplo:

```
PA_SetTextTileCol(1, i); // Cambia el color en la pantalla superior, valores de 0 a 9
PA_OutputSimpleText(1, 2, i, "Prueba de color..."); // La Pantalla 1 tiene diferentes colores
```

%cX La última forma que tenemos para establecer el color del texto, es usando *%cX* en una cadena de caracteres, siendo X el número de color *(0 - 9)*. La ventaja de éste sistema, es que puedes tener múltiples colores en la misma cadena de caracteres.

Aquí tienes un ejemplo:

```
PA_OutputText(0, 0, 0, Çolor test... %c1, %c2, %c3");
```

Fuentes personalizadas y borde del texto PAlib[6] te permite personalizar la fuente del texto y su borde. Necesitarás importar gráficos a tu proyecto, ya que tendrás que convertir algunos gráficos.

Custom font

Básicamente, una fuente customizada es una rejilla, cada cuadrado de está es un carácter específico. Aquí tienes una disposición básica:

Figura B.3: Rejilla de una fuente personalizada. Fuente: Palib.info

La disposición es muy importante, cada carácter tiene unas coordenadas en la rejilla. Un carácter tiene el mismo tamaño que cada cuadrado de la rejilla *(8 x 8 pixeles)*. Puedes editar esta disposición para hacer una propia.

Un ejemplo de incluir una fuente personalizada al código:

```
// Includes
#include <PA9.h> // Include for PA_Lib
#include "font/all_gfx.h"
#include "font/all_gfx.c"
// Function: main()
int main(int argc, char ** argv)
{
    PA_Init(); // Initializes PA_Lib
    PA_InitVBL(); // Initializes a standard VBL

    PA_InitText(0, 0); // Initialise the normal text on the bottom screen
    // Load a custom text font on the top screen
    PA_InitCustomText(1, //screen
        0, //background number
        newfont); //font name
    // PA_OutputSimpleText is the fastest text function, and displays 'static' text
    PA_OutputSimpleText(1, // screen
        2, // X postion
        2, // Y position
        "Hello World !!"); // display a custom text on the top screen
```

```
    PA_OutputSimpleText(0, 2, 2, "Hi there :p"); // and a normal one on the bottom
screen


    // Infinite loop to keep the program running
    while (1){
        PA_WaitForVBL();
    }

    return 0;
} // End of main()
```

Como puedes ver, usar una fuente propia es muy sencillo. Solo tienes q inicializar el texto con **PA_InitCustomText** en vez de **PA_InitText**.

B.4.4. Entrada

Existen dos tipos de entrada en DS: pad (todos los botones) y stylus (puntero). Además hay otras entradas, como por ejemplo un teclado implementado por PAlib[6] que se muestra en pantalla y se puede usar junto con el puntero de la videoconsola; también existe una característica de reconocimiento de formas usando lo que se conoce en la librería como **PA_Graffiti**, que permite reconocer figuras predefinidas o configurar nuevas; otro tipo de entrada de información sería el micrófono.

Pad

El estado del pad se almacena en una estructura homónima Pad. Dentro se puede diferenciar por tres tipos de eventos que definen los estados posibles de los botones: *Held*, *Released* y *Newpress*. *Held* es cuando una tecla está pulsada, permanecerá a en el valor uno mientras se encuentre en esa situación. *Released* se activará cuando el botón haya dejado de estar pulsado, permaneciendo en el valor uno durante un frame. Por último *NewPress* tendra el valor uno durante un frame cuando se pulse la tecla. Un ejemplo sencillo sería:

```
if(Pad.Held.Up){
    MoveUp();
}
if(Pad.Held.Down){
    MoveDown();
}
```

Cuyo significado en un hipotético juego podría ser, si está pulsada la dirección arriba, desplázate arriba, y si está pulsada la dirección abajo, desplázate hacia abajo.

Stylus

La lectura del Stylus es similar a la de Pad, existe una variable *Stylus* que almacena el estado del puntero y se actualiza cada frame. Se pueden consultar los mismos estados: *Held*, *Released* y *NewPress*. Además se puede conocer la posición del mismo consultando X e Y de la variable *Stylus*. Un ejemplo de código es el siguiente:

```
if (Stylus.Held){
    PA_SetSpriteXY(screen, sprite, Stylus.X, Stylus.Y);
}
```

Su significado es: si está el puntero pulsado sobre la pantalla, posiciona una imagen en la posición donde se encuentra el puntero.

Keyboard

Con PAlib[6] también se puede poner en pantalla un teclado para que los usuarios puedan escribir con el stylus en la pantalla inferior. PAlib[6] trae un teclado por defecto que podremos utilizar siempre. Para ello debemos utilizar un par de funciones para la carga y localización del teclado que trae por defecto.

Aquí podemos ver un ejemplo de cómo cargar el teclado en la pantalla inferior.

```
PA_Init(); // Inicia las PA_Lib
PA_InitVBL(); // Inicia un VBL estándar
PA_InitText(1, 0); // Inicia el sistema de texto
PA_InitKeyboard(2); // Carga el teclado en el background 2...
PA_KeyboardIn(20, 100); // Localización del teclado en la pantalla inferior

PA_OutputSimpleText(1, 7, 10, "Text : ");

s32 nletter = 0; // Numero de letras escritas
char letter = 0; // Siguiente letra a escribir.
char text[200]; // Este ha de ser nuestro texto.

// Loop infinito para mantener el programa en marcha
while (1)
{
    // Primero miraremos los cambios de colores, con A, B, Y, y X.
    if (Pad.Newpress.A) PA_SetKeyboardColor(0, 1); // Azul y rojo
```

```
    if (Pad.Newpress.B) PA_SetKeyboardColor(1, 0); // Rojo y azul
    if (Pad.Newpress.X) PA_SetKeyboardColor(2, 1); // Verde y rojo
    if (Pad.Newpress.Y) PA_SetKeyboardColor(0, 2); // Azul y verde
    letter = PA_CheckKeyboard();

    if (letter > 31) { // Aqui hay una nueva letra
        text[nletter] = letter;
        nletter++;
    }
    else if ((letter == PA_BACKSPACE)&&nletter) { // Espacio pulsado
        nletter–;
        text[nletter] = ' '; // Borra la última letra
    }
    else if (letter == '\n'){ // Enter pulsado
        text[nletter] = letter;
        nletter++;
    }

    PA_OutputSimpleText(1, 8, 11, text); // Escribe el texto
    PA_WaitForVBL();
}
```

- La primera función carga el teclado en la pantalla inferior, en el fondo número dos. También se puede cargar en cualquier otro fondo que se desee de los cuatro fondos disponibles. La segunda coloca el teclado en una posición en la pantalla.
- **PA_SetBgPalCol(Screen, BG, PA_RGB(31,31,31));** Es una función para cambiar el color de fondo. La pantalla debe estar entre cero o uno (cero para la pantalla inferior), BG es para el número del background (el del teclado) *(0 - 3)* y PA_RGB es el color *(0 - 31)*.
- **PA_SetKeyboardColor(0, 1);** Es una función para cambiar el color del teclado.
- **letter = PA_CheckKeyboard();** *CheckKeyboard* es un test para saber si has pulsado una tecla con el stylus, y devuelve la palabra que se está pulsando, o el valor cero si no está pulsado.

```
if (letter > 31) { // Esta es una nueva letra
    text[nletter] = letter;
    nletter++;
}
```

- Si la tecla es una letra, lo añade al texto sino la salta.
- **PA_OutputSimpleText(1, 8, 11, text);** Es la función para mostrar el texto que se escribe.

B.4.5. Sprites

Antes de comenzar es necesario conocer las características que trae DS. Es capaz de mostrar ciento veintiocho imágenes diferentes por cada pantalla, por tanto un total de doscientos cincuenta y seis diferentes. Cada imagen puede ser girada horizontal o verticalmente, desplazada por toda la pantalla, puede ser animada (actualizando la imagen de una secuencia), puede tener transparencias o incluso hacerse un mosaico. Las imágenes también pueden ser rotadas o aumentadas (o disminuidas). Existe una limitación y es que se pueden definir rotsets (rotaciones/aumentos) y aplicarlos a las imágenes, pero no se pueden definir más de treinta y dos rotsets diferentes por pantalla. De tal modo se podrán rotar/aumentar las imágenes que sean ya que se puede aplicar un mismo rotset a diferentes imágenes que se transformarán igual, pero sólo de treinta y dos maneras distintas.

Si a partir de ahora trabajaremos pintando imágenes en pantalla, hará falta indicar en qué posición de esta se va a mostrar, por tanto, será necesario conocer las posibles coordenadas de esta. Como se puede apreciar en la imagen:

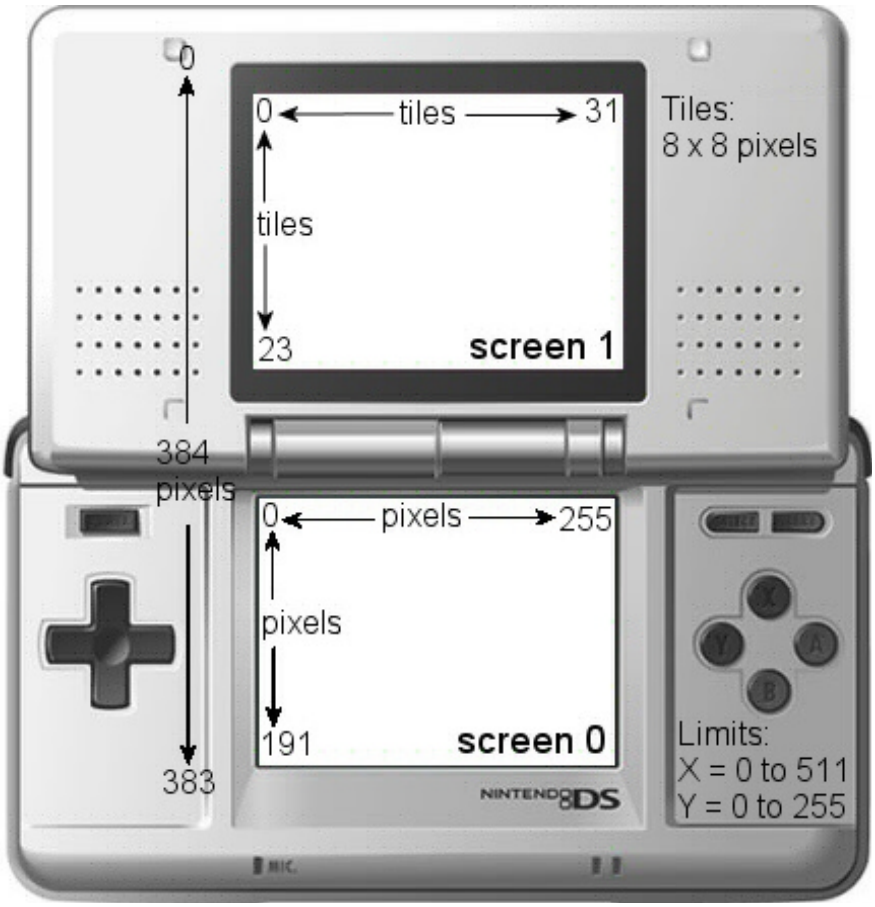


Figura B.4: Distribución pantalla Nintendo DS. Fuente: Palib.info

Como se puede apreciar, las coordenadas posibles en cada pantalla serán de 0 a 255 pixeles en ancho y de 0 a 191 pixeles en alto; siendo *(0, 0)* la posición más arriba a la izquierda y *(255,191)* la

más abajo a la derecha.

También es necesario conocer los tres diferentes modos de colores que existen en DS para los sprites:

- **Paleta de dieciséis colores**: muy usada en GBA.
- **Paleta de doscientos cincuenta y seis colores**: usa más memoria.
- **Imágenes de 16 bits**: Sin paleta, no se suele usar mucho.

Lo más común es usar paleta de doscientos cincuenta y seis colores ya que es el término medio con mejor rendimiento.

Además las imágenes deben tener unos tamaños específicos. En el caso de que la imagen que se vaya a usar tenga unas proporciones diferentes, habría que encapsularla a la siguiente superior que sea reconocida.

En muchas ocasiones ocurre que no interesa que la imagen se vea completamente, sino que sólo se muestre un contenido interior y el resto no se muestre, mostrándose hasta el perfil de la figura. Para esto será posible definir que un color en concreto sea transparente.

Mostrando sprites

El primer ejercicio es uno de los que trae la librería PAlib[6] para el que hará falta una imagen que trae en una de sus subcarpetas.

El código de la función main es:

```
#include <PA9.h>
// PAGfxConverter Include
#include "gfx/all_gfx.h"
#include "gfx/all_gfx.c"
int main(void){
    PA_Init(); //PAlib inicialización
    PA_InitVBL();
    PA_LoadSpritePal(
        0, // Pantalla
        0, // Número de paleta
        (void*)sprite0_Pal); // Nombre de paleta
    PA_CreateSprite(
        0, // Pantalla
        0, // Número de sprite
        (void*)vaisseau_Sprite, // Nombre de sprite
```

```
        OBJ_SIZE_32X32, // Tamaño de sprite
        1, // Modo de color 256
        0, // Número de paleta de sprite
        50, 50); // Posiciones X e Y en pantalla
    while(1) // Bucle infinito{
        PA_WaitForVBL();
    }
    return 0;
}
```

Las dos líneas de include, que no son para incluir la librería PAlib[6], tratan de cargar la imagen y la librería previamente convertidas a código C con la herramienta antes comentada PAGfx.

Después de la inicialización se llama a una nueva función **PA_LoadSpritePal** que carga la paleta de la imagen y su uso es el siguiente: **PA_LoadSpritePal (u8 screen, u8 palette_number, void *palette)**. En el caso del código escrito lo que hace es cargar una paleta en la pantalla número cero, en la primera paleta (la número cero) y la paleta cargar está en la dirección *sprite0_Pal*. La otra función que hace posible mostrar la imagen es **PA_CreateSprite (u8 screen, u8 obj_number, void *obj_data, u8 obj_shape, u8 obj_size, u8 color_mode, u8 palette, s16 x, s16 y)**. Lo que hace esta función en el ejemplo es lo siguiente: crea un sprite en la pantalla cero (la pantalla inferior donde ya está cargada la paleta, si no existiera esta correspondencia no se mostraría la imagen ya que cada pantalla maneja paletas diferentes), el segundo parámetro indica que el sprite se asociará con el índice cero (de los ciento veintiocho posibles, *0 - 127*) el cual lo representará para posibles futuras transformaciones del mismo, además de representar el nivel de profundidad al que se encuentra. Esto es que cuanto más pequeño es el valor del sprite, se encuentra por delante de los demás. El tercer parámetro es el puntero hacia la imagen previamente convertida, después se le indica el tamaño de la imagen (lo mejor es usar la macro como en el ejemplo que engloba los dos parámetros reales que hay que pasarle), a continuación se indica el modo de color siendo cero para dieciséis colores o uno para doscientos cincuenta y seis colores (normalmente será uno), después el número de paleta donde almacenamos la paleta correspondiente a esta imagen (en este caso el valor cero), para terminar se indica la coordenada donde se representará la imagen.

Desplazando sprites

PA_MoveSprite El siguiente código seguirá sin tener demasiada complejidad pero se trata de una tarea fundamental para poder realizar aplicaciones interactivas como es el caso de los videojuegos o aplicaciones gráficas.

```
// Los includes
#include <PA9.h>
// PAGfx Include
#include "gfx/all_gfx.c"
#include "gfx/all_gfx.h"
```

```
//Función principal
int main(void) {
    //Inicialización de PAlib
    PA_Init();
    PA_InitVBL();
    // Carga la paleta de imágenes
    PA_LoadSpritePal(
        0, // Pantalla
        0, // Número de paleta
        (void*)sprite0_Pal); // Nombre de paleta
        // Carga de 16 sprites
        u8 i = 0;
    for (i = 0; i < 16; i++)
        PA_CreateSprite( 0, i,(void*)vaisseau_Sprite, OBJ_SIZE_32X32,1, 0, i << 4, i
<< 3);
    while(1){
        // Usa la función PA_MoveSpite sobre todos los sprites
        for (i = 0; i < 16; i++) PA_MoveSprite(i);
            // La función PA_MoveSprite comprueba si está
            // Pulsado un sprite con el puntero, y se mueve con él.
            PA_WaitForVBL();
    }
    return 0;
}
```

El comienzo de este programa es muy parecido del anterior. Se inicializa PAlib y se carga la paleta de imágenes. Y ahora en lugar de crear un sprite se crean dieciséis haciendo uso de la misma imagen pero en posiciones diferentes.

El operador >> se utiliza en C para desplazar la cantidad de la izquierda el número de bits que se le indique en el operando derecho, lo que se corresponde con dividir entre potencias de dos la cantidad indicada. Si en cambio se usa el operador << la operación que se realiza es introducir un bit por la derecha, por tanto se multiplica la cantidad por potencias de dos. Como *i* va desde cero a quince lo que resulta es una línea diagonal descendente desde el eje de coordenadas en *(0, 0)* (hay que recordar que esto se corresponde con la posición arriba a la izquierda) hasta la coordenada *(15 x 16,15 x 8), (240,120)*.

Como nombre de referencia de cada sprite se le da el de la variable *i*, que como ya se ha dicho toma valores de cero a quince, siendo este el que se usará para hacer mover cada sprite junto con el puntero. Se trata de una tarea sencilla haciendo uso de la función **PA_MoveSprite (idsprite)**. Llamando a esta función e indicándole el identificador del sprite, este se desplazará si el puntero pasa por encima y se desvinculará cuando este se despegue.

PA_SetSpriteXY Con esta función habrá más libertad para desplazar los sprites a la coordenada que más convenga, en lugar de como sucedía con la función **PA_MoveSprite** que siempre se desplazaba al lugar donde estuviera el puntero. La función a utilizar será la siguiente **PA_SetSpriteXY (u8 screen, u8 sprite, s16 x, s16 y)**. Es necesario indicarle la pantalla donde se encuentra el sprite, el identificador de este y la coordenada X e Y a donde se quiera mover. Hay que tener en cuenta que la posición del sprite se corresponde con la esquina superior izquierda de este, no con su centro. Para recuperar las coordenadas de un sprite se usan las macros **PA_GetSpriteX(screen, obj) y PA_SetSpriteY(screen, obj, y)**.

Como ejemplo de uso, con este código el sprite se desplazará en función de la pulsación del cursor:

```
//Mover un sprite usando los cursores
#include <PA9.h>
// PAGfx Include
#include "gfx/all_gfx.c"
#include "gfx/all_gfx.h"
s32 x = 0; s32 y = 0; // posición del sprite
//Función main
int main(void){
    PA_Init(); //Inicialización de PAlib
    PA_InitVBL();
    PA_InitText(0,0);
    PA_LoadSpritePal( 0, // Pantalla
        0, // Número de paleta
        (void*)sprite0_Pal); // Nombre de paleta
        //Creación del sprite
    PA_CreateSprite(
        0, 0,(void*)vaisseau_Sprite, OBJ_SIZE_32X32,1, 0, 0, 0);
    while(1){ // Bucle principal
        // Actualiza la posición de acuerdo con la pulsación del cursor
        x += Pad.Held.Right - Pad.Held.Left;
        y += Pad.Held.Down - Pad.Held.Up;
        // Establece la posición del sprite
        PA_SetSpriteXY( 0, // pantalla
            0, // sprite
            x, // posición x
            y); // posición y
        PA_WaitForVBL();
    }
```

```
    return 0;
}
```

Dentro del bucle principal se actualizan dos variables x e y. La línea **x += Pad.Held.Right - Pad.Held.Left;** incrementará la variable X en uno si se ha pulsado hacia la derecha o se decrementará si se ha pulsado izquierda. En el caso de que ambos lados estuviesen pulsados la resta devolvería cero y se mantendría el valor de X. Lo mismo ocurre con la variable Y pero atendiendo a las pulsaciones de arriba y abajo. Una vez que se conoce la posición X e Y a la que debe desplazarse el sprite, se llama a la función **PA_SetSpriteXY** para moverlo.

Si, por ejemplo, se quisiera desplazar la imagen al lugar donde se pulse con el puntero, habría que cambiar la línea **PA_SetSpriteXY(0, 0, x, y);** por esta **PA_SetSpriteXY(0, 0, Stylus.X, Stylus.Y);**. Ya que las variables *Stylus.X* y *Stylus.Y* almacenan la coordenada donde se encuentra pulsado el puntero.

Lo siguiente será ver cómo se pueden aplicar transformaciones a las imágenes como por ejemplo rotaciones y zoom.

Rotaciones y zoom

Como se dijo anteriormente, los sprites pueden ser rotados y/o escalados, pero existe una pequeña limitación: todos los sprites pueden ser rotados, sin embargo, un sprite rotado necesita que se le aplique una rotación previamente definida llamada rotset y se pueden establecer hasta treinta y dos rotsets diferentes por pantalla, pudiendo aplicar un mismo rotset a diferentes sprites que presentarán la misma deformación. A continuación tres ejemplos: rotación, escalado y rotación/escalado.

Rotación Este es el código del ejemplo:

```
// Activa rotación de sprite y lo hace girar
#include <PA9.h>
// PAGfxConverter Include
#include "gfx/all_gfx.c"
#include "gfx/all_gfx.h"
int main(void){
    PA_Init();
    PA_InitVBL();
    // Carga la paleta
    PA_LoadSpritePal(
        0, // Pantalla
        0, // Número de paleta
        (void*)sprite0_Pal); // Nombre de paleta
    // Carga el sprite
```

```
        PA_CreateSprite(
            0, 0,(void*)vaisseau_Sprite, OBJ_SIZE_32X32,1, 0, 50, 50);
        // Activa las rotaciones para el sprite
        PA_SetSpriteRotEnable(
            0,// pantalla
            0,// Número de sprite
            0);// Número de rotset. Hay 32 (0-31) por pantalla
        u16 angle = 0; // Ángulo de rotación...
        while(1){
            ++angle; // modifica el ángulo
            // Limita el rango de 0-511.
            // Funciona sólo con 1, 3, 7, 15, 31... (2^n - 1)
            angle &= 511;
            // Función para rotaciones sin zoom
            PA_SetRotsetNoZoom(
                0, //pantalla
                0, // rotset
                angle); // ángulo, de 0 a 511
            PA_WaitForVBL(); // Sincronización
        }
        return 0;
    }
```

La primera función nueva que aparece es **PA_SetSpriteRotEnable(screen, sprite, rotset)**. Con esta función se vincula una sprite existente en una pantalla con un rotset. A partir de esta llamada el sprite está listo para ser transformado, si se quisiera deshabilitar esta opción habría que usarse la función **PA_SetSpriteRotDisable(screen, sprite)**. Después se calcula el valor del ángulo de rotación que se le aplicará al sprite. Como se puede ver no se trata de un rango de ángulos normal de trescientos sesenta grados, sino que la vuelta completa cubre las posiciones que van desde cero a quinientos once. La razón es porque de este modo es mucho más eficiente la Nintendo DS que con los ángulos normales. También indicar que el giro se produce en sentido contrario a las agujas del reloj.

Por último, se realiza la transformación haciendo uso de la función **PA_SetRotsetNoZoom (u8 screen, u8 rotset, s16 angle)**. De este modo se modifica el rotset, como se puede apreciar no se hace referencia al id del sprite ya que este se vinculó anteriormente y al modificar el rotset ya se aplica la transformación al sprite. Hay que indicar en cual pantalla se encuentra el rotset, su id *(0 - 31)* y el ángulo de giro *(0 - 511)*. El giro del sprite se realiza desde su punto central.

Zoom Código que muestra imágenes rotando:

```
// Activa zoom de sprite
#include <PA9.h>
// PAGfxConverter Include
#include "gfx/all_gfx.c"
#include "gfx/all_gfx.h"
int main(void){
    PA_Init();
    PA_InitVBL();
    // Carga la paleta
    PA_LoadSpritePal(
        0, // Pantalla
        0, // Número de paleta
        (void*)sprite0_Pal); // Nombre de paleta
    // Carga el sprite
    PA_CreateSprite(
        0, 0,(void*)vaisseau_Sprite, OBJ_SIZE_32X32,1, 0, 50, 50);
    // Habilita doble tamaño, lo cual significa que el sprite puede ser más grande que su
tamaño normal
    PA_SetSpriteDblsize(0, 0, 1);
    // Este es otro sprite sin doble tamaño para mostrar la diferencia
    PA_CreateSprite(
        0, 1,(void*)vaisseau_Sprite, OBJ_SIZE_32X32,1, 0, 120, 66);
    // Activa las rotaciones para el sprite
    PA_SetSpriteRotEnable(
        0,// pantalla
        0,// Número de sprite
        0);// Número de rotset. Hay 32 (0-31) por pantalla
    // Mismo rotset para el otro sprite, se le aplicará la misma transformación
    PA_SetSpriteRotEnable(0, 1, 0);
    // Zoom. 256 significa no zoom, 512 es el doble de pequeño, 128 es el doble de grande
    u16 zoom = 256;
    while(1){
        // Modifica el ángulo según las teclas
        zoom -= Pad.Held.Up - Pad.Held.Down;
        // Función para zoom sin rotaciones
```

```
            PA_SetRotsetNoAngle(
                0, // pantalla
                0, // rotset
                zoom, zoom); // Horizontal zoom, Vertical zoom
            PA_WaitForVBL(); // Sincronización
        }
        return 0;
    }
```

Ejemplo muy similar al anterior debido a que un rotset puede representar tanto zoom como rotaciones, por tanto sólo hay que modificar el tipo de transformación que se aplica a la imagen. Sin embargo, es necesario comentar algunos detalles que no aparecían anteriormente. Ahora usamos una variable llamada zoom que se va modificando para aplicar un zoom distinto en cada ocasión. Se inicializa con doscientos cincuenta y seis ya que esta cifra se corresponde al tamaño normal, es decir 100 % de su tamaño. Si la cifra es mayor se le aplicará una reducción, por ejemplo quinientos doce se corresponde con el 50 % del tamaño original. Por consiguiente, si la cifra es inferior se aumenta el tamaño, con un valor de ciento veintiocho la imagen se muestra al 200 % de su tamaño.

Lo siguiente es aplicar la transformación, como ocurría en el ejemplo anterior esto es modificando las propiedades del rotset que está vinculado al sprite. En este caso usaremos una función que permite modificar exclusivamente el zoom, obviando la transformación de rotación. Esta función es **PA_SetRotsetNoAngle (u8 screen, u8 rotset, u16 zoomx, u16 zoomy)** que permite aplicar un zoom horizontal y otro vertical por separado. En este código se ha aplicado un mismo rotset a dos imágenes y como se puede observar, con hacer una sola llamada a esta función se modifica el rotset, que a su vez se observa como el zoom se aplica a ambas imágenes.

El problema de aumentar una imagen es que si esta ocupa por ejemplo *32 x 32 píxeles*, al hacer zoom se pasará de sus límites y se saldrá de los límites que tiene establecidos. Por ese motivo existe la función **PA_SetSpriteDblsize(screen, obj, dblsize)**, esta permite habilitar la opción para que la imagen pueda abarcar el doble de su tamaño. El efecto se puede comprobar en el ejemplo ya que esta función se ha aplicado solamente a una de las imágenes.

Zoom y rotación Para esta ocasión no considero necesario copiar aquí ningún ejemplo aunque si irá incluido dentro de los códigos fuentes por si resulta necesario. La función usada para realizar estas dos operaciones es **PA_SetRotset (u8 screen, u8 rotset, s16 angle, u16 zoomx, u16 zoomy)**, que consiste en una combinación de las dos vistas anteriormente donde se puede especificar el ángulo de rotación y el zoom horizontal y vertical del rotset.

Volteo de imágenes Se trata de conseguir que una imagen se vea reflejada bien horizontalmente o bien verticalmente. Las funciones para conseguir son **PA_SetSpriteHflip(screen, obj, hflip)** y **PA_SetSpriteVflip(screen, obj, vflip)** respectivamente. Lo único que hay que hacer es indicar en el tercer parámetro con valor cero ó uno, si se quiere realizar la acción. El ejemplo que ilustra cómo hacer esta funcionalidad es el siguiente:

```
#include <PA9.h>
// PAGfxConverter Include
#include "gfx/all_gfx.c"
#include "gfx/all_gfx.h"
int main(void){
    PA_Init(); //Inicialización de PAlib
    PA_InitVBL();
    PA_LoadSpritePal(
        0, // Pantalla
        0, // Número de paleta
        (void*)sprite0_Pal); // Nombre de paleta
    // Se crearán cuatro sprites que serán volteados
    // Cada uno de diferente manera
    PA_CreateSprite(
        0, 0, (void*)mollusk_Sprite, OBJ_SIZE_32X32, 1, 0, 0, 50);
    PA_CreateSprite(
        0, 1, (void*)mollusk_Sprite, OBJ_SIZE_32X32, 1, 0, 64, 50);
    PA_CreateSprite(
        0, 2, (void*)mollusk_Sprite, OBJ_SIZE_32X32, 1, 0, 128, 50);
    PA_CreateSprite(
        0, 3, (void*)mollusk_Sprite, OBJ_SIZE_32X32, 1, 0, 192, 50);
    // Flip para los sprites 1 a 3
    PA_SetSpriteHflip(0, 1, 1); // HFlip -> Horizontal flip
    PA_SetSpriteVflip(0, 2, 1); // VFlip -> Vertical flip
    // Horizontal y Vertical flip
    PA_SetSpriteHflip(0, 3, 1); PA_SetSpriteVflip(0, 3, 1);
    while(1) // Bucle infinito{
    }
    return 0;
}
```

Transparencias

También conocido como alpha-blending. PAlib[6] usa el hardware de Nintendo DS para llevar a cabo esta tarea. Existe una limitación y es que se puede aplicar la transparencia a todas las imágenes que sean, pero sólo se podrá disponer de un grado de transparencia, es decir, que todas las imágenes a las que le afecte esta modificación se verán con la misma transparencia.

Lo primero que hay que realizar para poder dar transparencia a un sprite es habilitar dicha opción con **PA_SetSpriteMode(screen, sprite, obj_mode)**. En el tercer parámetro se establece el modo siendo el valor cero para modo normal, uno para transparencia y dos para modo ventana.

Una vez activado, el sprite no se mostrará transparentado todavía. Antes es necesario activar el sistema de efectos especiales para DS, estableciendo el modo de transparencia de nuevo con **"PA_EnableSpecialFx (Screen, SFX_ALPHA (Alpha blending mode), 0, SFX_BG0 | SFX_BG1 | SFX_BG2 | SFX_BG3 | SFX_BD)"**. Por último se establece el nivel de alpha al sprite con "**PA_SetSFXAlpha**".

Comentar además que este ejemplo no funciona bien con los emuladores que he probado, es necesario hacerlo funcionar en el hardware de la videoconsola Nintendo DS. El código de ejemplo es el siguiente:

```
#include <PA9.h>
// PAGfxConverter Include
#include "gfx/all_gfx.c"
#include "gfx/all_gfx.h"
int main(void){
    PA_Init(); //Inicialización de PAlib
    PA_InitVBL();
    PA_LoadSpritePal(
        0, // Pantalla
        0, // Número de paleta
        (void*)sprite0_Pal); // Nombre de paleta
    // Se crearán dos sprites
    PA_CreateSprite(
        0, 0, (void*)mollusk_Sprite, OBJ_SIZE_32X32, 1, 0, 0, 50);
    PA_CreateSprite(
        0, 1, (void*)mollusk_Sprite, OBJ_SIZE_32X32, 1, 0, 64, 50);
    PA_SetSpriteMode(
        0, // Pantalla
        0, // Sprite
        1); // Alphablending
    s16 alpha = 7; // Nivel de transparencia
    // Habilita el alpha-blending
    PA_EnableSpecialFx(
        0, // Pantalla
        SFX_ALPHA, // Modo alpha blending
        0, // Nada
        SFX_BG0 | SFX_BG1 | SFX_BG2 | SFX_BG3 | SFX_BD); // Lo normal
    while(1){
```

```
            alpha += Pad.Newpress.Up - Pad.Newpress.Down;
            PA_SetSFXAlpha(
                0, // Pantalla
                alpha, // Nivel de alpha, 0-15
                15); // Dejar a 15
            PA_WaitForVBL();
        }
        return 0;
    }
```

Profundidad

Este es también un aspecto importante ya que si existen varias imágenes superpuestas, con el concepto de profundidad se podrá establecer estar prioridad.

Existen dos tipos de prioridades en DS:

- **Prioridad de sprite**: una imagen con un número de sprite menor se verá por encima de otro sprite con un número mayor.
- **Prioridad de fondo**: Está por encima de la prioridad de sprite. Por defecto todos los sprites están en el fondo número cero. Se puede establecer que un sprite se sitúe en frente de otro fondo, *0 - 3*. Una imagen puesta en el fondo número dos estará detrás de todos los sprites con un fondo de prioridad de valor cero ó uno, y por delante de todos los sprites con un fondo de prioridad tres.

Para establecer prioridades se usa la función "**PA_SetSpritePrio(screen, obj, prio)**".

Animaciones

Un sprite puede tener varias animaciones que van cambiando según van pasando los frames del juego. Sobre todo se usa para darle movimiento a los sprites o realizar ciertas acciones para que el usuario vea que se mueven según va transcurriendo el tiempo. Estas animaciones son principalmente para gráficos en dos dimensiones.

En PAlib[6] se deben de colocar el sprite con todas sus animaciones en una columna completa donde cada sprite será una posición en la animación, y deberán tener todos los sprites el mismo tamaño para que, cuando cargue el sprite en cuestión siempre este en la misma posición.

Un ejemplo de animación seria:

Figura B.5: Ejemplo de la animación de una explosión. Fuente: Palib.info

Para colocar un sprite en una animación concreta usamos la función **PA_SetSpriteAnim(pantalla, sprite, número de frame)**, y el sprite estará siempre en la posición que le indiquemos en cada momento. El número en el frame dependerá de la posición vertical que se encuentre la animación.

También podemos añadir animaciones automáticas que se irán ejecutando con el tiempo sin necesidad de controlarlo. Estas funciones son:

> PA_StartSpriteAnim(pantalla, sprite, primer frame, último frame, velocidad (en frames por segundo))

Una vez ejecutada la función anima el sprite en la dirección seleccionada, a la velocidad seleccionada y para detenerlo podemos usar **PA_StopSpriteAnim(pantalla, sprite)** o **PA_SpriteAnimPause(pantalla,sprite, pausa (valor uno para pausar, cero para reanudar))**.

Un ejemplo de animación podría ser:

```
PA_CreateSprite(0, 2,(void*)explosion_Sprite, OBJ_SIZE_64X64,1, 0, 128, 64);
PA_CreateSprite(0, 3,(void*)explosion_Sprite, OBJ_SIZE_64X64,1, 0, 196, 64);
```

```
// Comenzar la animación.
PA_StartSpriteAnim(0, // pantalla
        0, // número de sprite
        0, // el primer frame es el 0
        6, // el último frame es el 6
        5); // Velocidad, establecida a 5 frames por segundo.
PA_StartSpriteAnim(0, 1, 0, 6, 15); //la velocidad a 15 fps.
PA_StartSpriteAnim(0, 2, 0, 6, 30);
PA_StartSpriteAnim(0, 3, 0, 6, 60);
```

B.4.6. Backgrounds

Las backgrounds son los fondos que se colocan al final de la pantalla para crear los escenarios, tanto de los juegos, como de las aplicaciones que necesitemos un fondo determinado o la creación de menús.

- La DS puede mostrar cuatro fondos diferentes en cada pantalla, cada uno con su propia paleta de doscientos cincuenta y seis colores. Cada fondo *tileado* se compone de dos partes:
  - Un tileset, que es un conjunto de tiles de *8 x 8 píxels*, bloques que componen la imagen global.
  - Un mapa de tiles, que es un pequeño mapa que puede tener de *256 x 256* a *512 x 512 píxels*, que describe como están alineados los tiles en la pantalla.
- Se puede sustituir un mapa tileado por un fondo de *8 o 16 bits*, en el que puedes cargar un mapa de bit, un bmp, un gif, o incluso un jpg. Sin embargo, este tipo de fondo está limitado a uno por pantalla, y ocupa mucha RAM.

La DS puede también mostrar hasta dos fondos rotándose o escalándose por pantalla.

Carga

Para la carga de un fondo, utilizaremos la función global de carga de cualquier background llamada **PA_Background(pantalla, Número fondo, Imagen)**. Esta función carga en la localización indicada la imagen pasada por referencia. Hay que tener en cuenta que cada pantalla solo puede tener cuatro fondos simultáneos, así que si queremos cargar un quinto hay que eliminar uno de los ya cargados.

Desplazamiento de fondos (Scrolling)

El scrolling o desplazamiento del fondo es simplemente mover el fondo lateralmente o de arriba a abajo para un fin. Se puede utilizar este método para crear escenarios en los juegos o fondos que se

mueven constantemente. Las funciones utilizadas para este efecto se pueden describir en el siguiente ejemplo:

```
PA_InitText(1, 0);
// Cargar los fondos y sus paletas...
PA_Background(0, 3, BG3);
PA_Background(1, 3, BG3);

s32 scrollx = 0;
s32 scrolly = 0;
// Bucle infinito
while (1){
    // Modificamos scrollx y scrolly según los botones pulsados
    scrollx += (Pad.Held.Left - Pad.Held.Right) * 4; // Mover horizontalmente 4 pixels por pulsación
    scrolly += (Pad.Held.Up - Pad.Held.Down) * 4; // Mover verticalmente 4 pixels por pulsación
    // Desplazar el fondo a scrollx, scrolly...
    PA_EasyBgScrollXY(0, // pantalla
            3, // Número de fondo
            scrollx, // desplazamiento horizontal
            scrolly); // desplazamiento vertical

    // Mostrar los desplazamientos horizontal y vertical :
    PA_OutputText(1, 0, 0, "x :%d \ny :%d ", scrollx, scrolly);

    PA_WaitForVBL();
}
```

También podemos realizar scroll horizontal o vertical únicamente utilizando las funciones:

- **PA_EasyBgScrollX**
- **PA_EasyBgScrollY**

Scroll Parallax

El Scroll Parallax (Parallax Scrolling) es una técnica que permite distinta velocidad de desplazamiento en los fondos, causando una impresión de profundidad y en cierto modo efecto de tridimensionalidad.

PAlib[6] tiene algunas funciones para scroll parallax, una para inicializarlo, y una para usarlo.

```
// Cargar los 3 fondos en la pantalla inferior...
PA_EasyBgLoad(0, 1, BG1);      //pantalla, número de fondo, nombre de fondo
PA_EasyBgLoad(0, 2, BG2);
PA_EasyBgLoad(0, 3, BG3);
// Inicializar el parallax verticalmente (eje Y) para ambos fondos
// 256 es la velocidad normal, 128 la mitad de velocidad, 512 el doble...
PA_InitParallaxY(0, //pantalla
                 0, //Velocidad del Parallax para el fondo 0. 0 significa que no hay parallax
                 256, // Velocidad normal para el fondo 1
                 192, // Velocidad de 3/4 para el fondo 2
                 128); // Mitad de velocidad para el 3
PA_InitParallaxY(1, 0, 256, 192, 128);


s32 scroll = 0;


// bucle infinito
while (1)
{
    scroll += 1; // desplazar un pixel.
    // Los fondos con velocidad parallax de 256 se desplazarán 1 pixel, 192 desplazará
0.75, y 128 desplazará 0.5
    // También podríamos haber puesto una velocidad negativa para tener algunos fondos
desplazándose en dirección contraria


    PA_ParallaxScrollY(0, -scroll); // Desplazar los fondos de la pantalla 0.
    PA_WaitForVBL();
}
```

Fondos de 8/16 bit

La principal diferencia entre los fondos de *8/16bit* y los otros fondos que hemos visto es que son dibujados pixel a pixel.

Ventajas e inconvenientes de usar fondos de 8/16bit Inconvenientes

- Ocupan mucho más memoria de vídeo (VRAM) que los otros fondos. Un fondo de *8 bits* ocupa hasta tres octavas partes de la memoria RAM de una pantalla (37.5 %), y uno de *16 bit* ocupa una seis octavas partes (75 %).
- Son muy lentos de dibujar, por dibujarse pixel por pixel, así que tarda como sesenta y cuatro veces más cambiar un bloque de *8 x 8* en un fondo de mapa de bits que en uno tileado. Además, los fondos de *8 bits* tienen limitaciones específicas que los hacen incluso más lentos que los de *16 bits* dibujando pixel por pixel.
- Similar para el borrado de los fondos, toma mucho tiempo, mientras que es bastante instantáneo para los otros tipos de fondo.
- Solo se pueden poner en el fondo número tres. No es realmente un problema, ya que puedes cambiar la prioridad del fondo.

Ventajas

- Puedes dibujar directamente en la pantalla. PAlib[6] ofrece funciones para dibujar usando el stylus.
- Puedes cargar ciertos formatos de imagen directamente en un fondo de *8 bit* o *16 bit* (dependiendo del formato) sin tener que convertirla. A pesar de que esto es lento, tiene la ventaja de que estas imágenes a menudo son bastante pequeñas para incluirlas en la ROM, así que es ideal para 'splash screens' o fondos estáticos que solo necesitan cargarse una vez, y para los cuales la velocidad no importa. Los formatos soportados son jpeg (*16 bits* solo), gif (*8/16 bits*), bmp (*16 bits*), y raw (*8/16 bits*).
- Los fondos de *16 bits* no hay limitación de paleta, y todos los colores posibles en la pantalla. Esto es estupendo, usado en conjunción con jpegs, por ejemplo.

Para crear un fondo de *8 o 16 bits* primero debes crear la imagen en cualquier programa de dibujo que uses. Asegúrate que la imagen es de *256 x 192* o sino la imagen aparecerá modificada en la pantalla. Si es un fondo de *16 bits* pon el nombre del fichero, *16 bit*, y ningún nombre de paleta. Si es de *8 bits*, pon el nombre del fichero, *8 bits*, y el nombre de la paleta.

Para usar fondos de *8 bits* en el programa usamos el siguiente ejemplo:

```
PA_Init8bitBg(0,3);
PA_Load8bitBgPal(0,(void*)bmp_Pal);
PA_Load8bitBitmap(0,background_Bitmap);
```

PA_Init8bitBg(0,3); inicializa la pantalla número cero para poner un fondo de *8 bits* con una prioridad de número tres. La prioridad puede ser cualquier valor de cero a tres.

PA_Load8bitBgPal(0,(void*)bmp_Pal); carga la paleta para el mapa de bits de *8 bits* que vas a usar finalmente **PA_Load8bitBitmap(0,background_Bitmap);** pone la imagen en la pantalla. el número cero es el número de la pantalla y *background_Bitmap* es el nombre del archivo seguido de "_Bitmap".

B.4.7. Funciones Matemáticas

Números aleatorios

Los números aleatorios hacen falta en muchas ocasiones ya que permiten conseguir comportamientos que parecen no definidos, por ejemplo, son muy útiles en algoritmos de inteligencia artificial. A continuación se explican las funciones que generan estos números aleatorios.

PA_Rand() Esta es la función más sencilla, genera un número aleatorio pero de un tamaño muy grande. Las siguientes funciones evitarán tener que usar esta función y posteriores transformaciones sobre su resultado.

PA_RandMax(max) A esta función se le puede pasar un valor máximo de manera que si por ejemplo se le pasa el valor de seis, devolverá un valor aleatorio entre los valores cero y seis, ambos inclusive.

PA_RandMinMax(min,max) Con **PA_RanMinMax** se puede establecer un valor mínimo y otro máximo inclusive para el valor devuelto.

PA_InitRand() Llamando a esta función se podrá inicializar la semilla de números aleatorios con la hora y fecha en el momento de su invocación. Usándola una vez antes del bucle principal, se podrán conseguir que cada ejecución de la aplicación genere números distintos con mayor probabilidad.

Decimales con punto fijo

Este apartado explica un punto importante en relación con el rendimiento. Algo que siempre penaliza mucho las operaciones realizadas por el procesador son los cálculos con números decimales. Esto puede tener mayor repercusión incluso en sistemas con potencia limitada, como es el caso de una videoconsola portátil. La DS es muy lenta trabajando con tipos de coma flotante como float. Sin embargo, evitar usarlos no es una solución en muchas ocasiones que es necesario una dimensión real.

Usando *s32*, variable de *32 bit*, nosotros podremos reservar los últimos *8 bits* para almacenar la parte decimal y los veinticuatro restantes para la parte entera. Los *8 bits* pueden tener valores entre cero y doscientos cincuenta y cinco, por tanto la precisión que se puede llegar a tener es de *1/256*.

El código es el siguiente:

```
#include <PA9.h>
// PAGfxConverter Include
#include "gfx/all_gfx.c"
#include "gfx/all_gfx.h"
int main(void){
```

```
    PA_Init(); //Inicialización PAlib
    PA_InitVBL();
    PA_InitText(0, 0);
    s32 speed1 = 256; // Velocidad 1
    PA_OutputText(0, 15, 2, "1 pixel/frame");
    s32 speed2 = 128; // Velocidad 0.5
    PA_OutputText(0, 15, 10, "0.5 pixel/frame");
    s32 speed3 = 64; // Velocidad 0.25
    PA_OutputText(0, 15, 18, "0.25 pixel/frame");
    PA_LoadSpritePal(0, 0, (void*)sprite0_Pal);
    PA_CreateSprite(
        0, 0, (void*)vaisseau_Sprite, OBJ_SIZE_32X32, 1, 0, 0, 0);
    PA_CreateSprite(
        0, 1, (void*)vaisseau_Sprite, OBJ_SIZE_32X32, 1, 0, 0, 64);
    PA_CreateSprite(
        0, 2, (void*)vaisseau_Sprite, OBJ_SIZE_32X32, 1, 0, 0, 128);
    // Todos las imágenes comienzan a la izquierda
    s32 spritex1 = 0; s32 spritex2 = 0; s32 spritex3 = 0;
    while(1){
        // Cada imagen se mueve con su velocidad correspondiente
        spritex1 += speed1; spritex2 += speed2; spritex3 += speed3;
        // Posiciona todos los sprites, >>8 lo devuelve a
        // Su posición normal
        PA_SetSpriteX(0, 0, spritex1>>8);
        PA_SetSpriteX(0, 1, spritex2>>8);
        PA_SetSpriteX(0, 2, spritex3>>8);
        PA_WaitForVBL();
    }
    return 0;
}
```

Se observan las tres variables de tipo *s32* que controlan las velocidades de las imágenes:

speed1, *speed2* y *speed3*. Sus valores son *256*, *128* y *64* respectivamente que se corresponden con velocidades de *1, 0.5 y 0.25* píxeles por frame. Después dentro del bucle principal se actualizan las posiciones en X según la velocidad que tiene cada imagen. Por último se posicionan los sprites realizando una conversión.

Trayectorias y ángulos

En esta sección se va a explicar cómo se usan los ángulos en PAlib[6], como aplicar funciones trigonométricas como senos y cosenos, y después se verá un ejemplo.

Ángulos en PAlib Algo a lo que se ha hecho mención anteriormente es que los ángulos en PAlib[6] no son como los cotidianos. El rango en DS va desde cero grados hasta quinientos once grados y en sentido contrario a las agujas del reloj. El orden es el siguiente, cero grados es el lado derecho, ciento veintiocho apunta hacia arriba, doscientos cincuenta y seis es el lado izquierdo y trescientos ochenta y cuatro se corresponde con abajo. Se puede ver ilustrado en la figura.

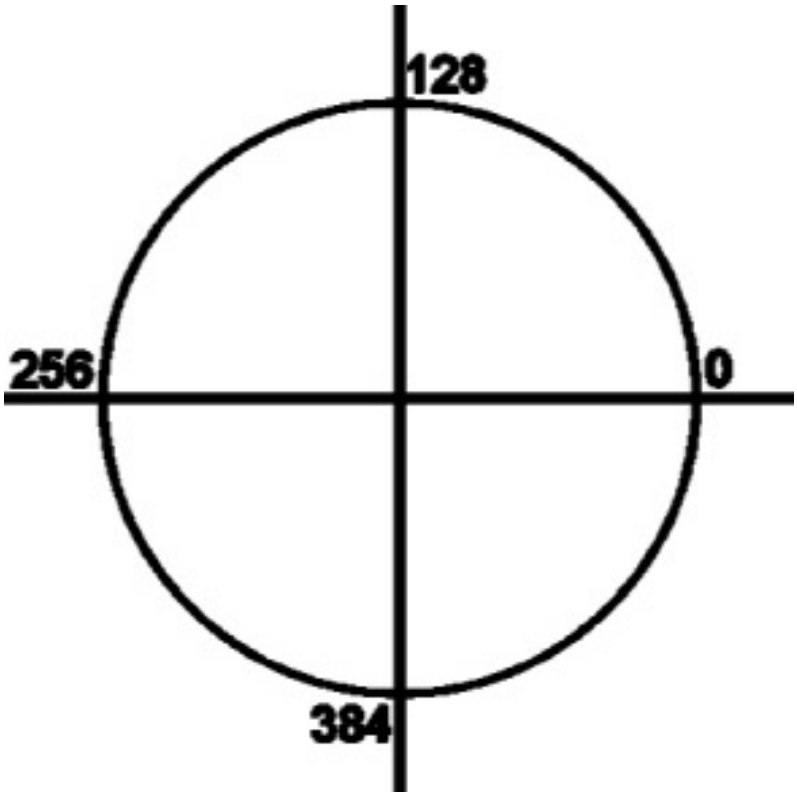


Figura B.6: Ángulos con PAlib. Fuente: Palib.info

PAlib[6] ofrece una función que simplifica el trabajar con estos ángulos, se trata de **PA_GetAngle (s32 startx, s32 starty, s32 targetx, s32 targety)** y devuelve el ángulo formado entre la línea horizontal y la línea compuesta por los dos puntos (startx, starty) y (targetx, targety).

El código para probar esta función es el siguiente:

```
// Includes
#include <PA9.h>
```

```
#include "gfx/vaisseau.raw.c"
#include "gfx/master.pal.c"
int main(void){
    PA_Init();
    PA_InitVBL();
    PA_InitText(1,0); // Pantalla superior
    PA_LoadSpritePal(0, 0, (void*)master_Palette);
    PA_CreateSprite(
        0, 0,(void*)vaisseau_Bitmap,
        OBJ_SIZE_32X32,1, 0, 128-16, 96-16);
    while(1){
        PA_OutputText(1, 5, 10, .Ángulo :%d ", PA_GetAngle(128, 96, Stylus.X, Sty-
lus.Y));
        PA_WaitForVBL();
    }
    return 0;
}
```

Muestra en la pantalla inferior una imagen de una nave y en la pantalla superior se muestra el grado que existe entre la línea horizontal, que atraviesa la nave, y la línea que se forma desde el centro de la nave con la pulsación del puntero sobre la pantalla.

Senos y Cosenos La primera diferencia entre estas funciones es que no devuelven un valor entre menos uno y uno, sino entre el valos menos doscientos cincuenta y seis y doscientos cincuenta y seis. De esta manera es mucho más eficiente ya que hace uso de números decimales con punto fijo. A continuación se muestra una imagen que representa esta situación:

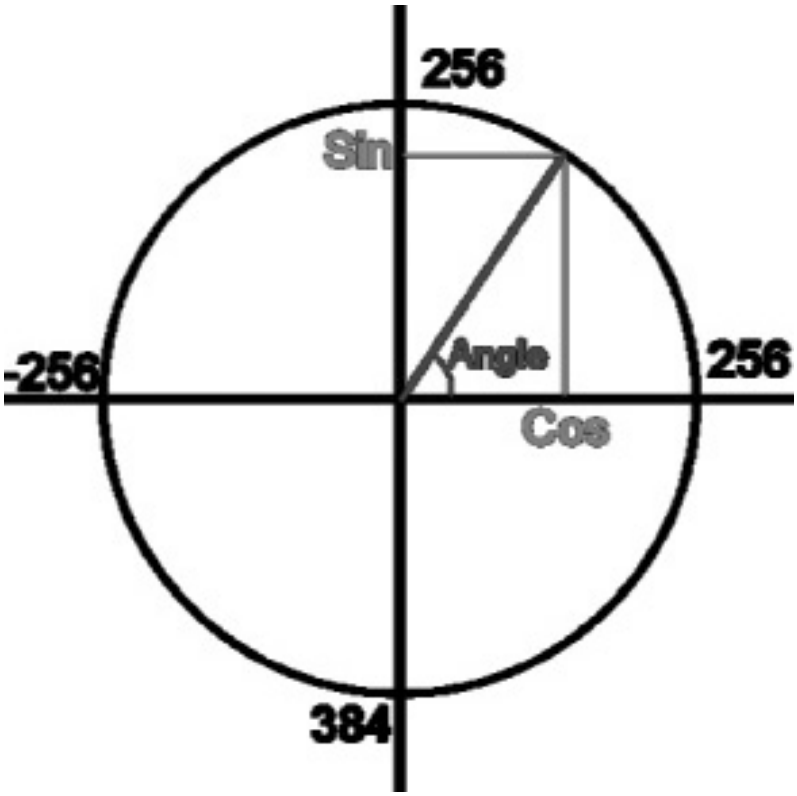


Figura B.7: Seno y coseno con PAlib. Fuente: Palib.info

Para obtener el seno y el coseno a partir de un ángulo se hace uso de las funciones **PA_Sin(angle)** y **PA_Cos(angle)** respectivamente.

Ejemplo de trayectoria Este ejemplo muestra una nave en el centro de la pantalla que puede desplazarse hacia delante, atrás y girar a derecha e izquierda. Aquí está el código:

```
// Includes
#include <PA9.h>
// PAGfxConverter Include
#include "gfx/all_gfx.c"
#include "gfx/all_gfx.h"
int main(void){
    PA_Init();
    PA_InitVBL();
    PA_InitText(1,0); // Pantalla superior
    PA_LoadSpritePal(0, 0, (void*)sprite0_Pal);
    // Crea la nave en el centro
```

```
        PA_CreateSprite(
            0, 0,(void*)vaisseau_Sprite,
            OBJ_SIZE_32X32,1, 0, 128-16, 96-16);
        // Habilita rotaciones y establece el Rotset 0
        PA_SetSpriteRotEnable(0,0,0);
        // Posición x de la nave en 8bit con punto fijo
        s32 x = (128-16) << 8;
        // Posición Y
        s32 y = (96-16) << 8;
        // Dirección a la que mover
        u16 angle = 0;
        while(1){
            angle += Pad.Held.Left - Pad.Held.Right;
            // Gira la nave en la dirección correcta
            PA_SetRotsetNoZoom(0, 0, angle);
            if (Pad.Held.Up){ // Mueve hacia delante
                x += PA_Cos(angle);
                y -= PA_Sin(angle);
            }
            if (Pad.Held.Down){ // Mueve hacia atrás
                x += -PA_Cos(angle);
                y -= -PA_Sin(angle);
            }
            PA_OutputText(1, 5, 10, .Ángulo :%d ", angle);
            // Posición de imagen convertida a normal
            PA_SetSpriteXY(0, 0, x>>8, y>>8);
            PA_WaitForVBL();
        }
        return 0;
    }
```

Primero se crea una imagen y se le vincula a un rotset para poder girarla. Después se inicializan las variables que se usarán, en este caso las posiciones X e Y, y el ángulo de dirección. Las posiciones X e Y son variables de tipo *s32* por tanto estamos usando decimales con punto fijo, por lo que hay que hacer uso de la operación *<< 8*.

Dentro del bucle lo primero que se actualiza es el ángulo, incrementándose si se gira a la izquierda o decrementándose si es a la derecha (sentido inverso a las agujas del reloj). Después se actualiza la transformación del rotset para girar el sprite. Por último es necesario actualizar las coordenadas de

posición. A la posición X se le añade el coseno del ángulo, mientras que a la posición Y es necesario restarle el seno del ángulo ya que la parte superior de la pantalla tiene $y = 0$ lo contrario a los ejes de coordenadas habituales que las posiciones bajas de Y están en la parte inferior, por eso es necesario restar. Finalmente se actualiza la posición con la función **PA_SetSpriteXY** teniendo en cuenta usar $>> 8$ para contrarrestar la operación $<< 8$ usada en la inicialización de las variables.

B.4.8. Sonido

El sonido se presenta como una parte importante en cualquier aplicación gráfica ya que sin sonido quedaría incompleta quedando por muy cuidados que estuvieran el resto de detalles. Ya sea desde acompañar el juego por una música que revele el estado de las situaciones sumergiendo al usuario en el ambiente de lo que está viendo, o bien ayudando a la trama con efectos de sonidos que acompañan a las acciones o eventos que se produzcan. Tal como lo representa la palabra audiovisual, la parte gráfica junto con el sonido se complementan la una con la otra para conseguir un mismo objetivo, transmitir sensaciones al usuario. Por tanto, se convierte el audio en una parte imprescindible que no puede dejar de mencionarse.

En PAlib[6] existen dos métodos diferentes para cargar sonidos en la salida de audio: el formato raw para reproducir sonido wav, usado sobre todo para efectos especiales debido a que ocupa mucho espacio; y la reproducción mod que es perfecta para sonido de fondo en bucle ya que ocupa muy poco espacio.

Archivos de sonido Raw

La DS no puede reproducir archivos wav ni otros formatos conocidos como mp3 u ogg, hace falta convertir los audios a un formato raw para luego reproducirlos.

La conversión a formato raw debe llevarse a cabo usando algún programa. Uno de los más sencillos es Switch que en su versión gratuita permite la conversión entre varios formatos incluido raw. Basándome en este convertidor, además de seleccionar un formato de salida .raw, es posible especificar algunos criterios de codificación. Para obtener archivos de poco tamaño con una calidad aceptable se pueden dejar estas opciones:

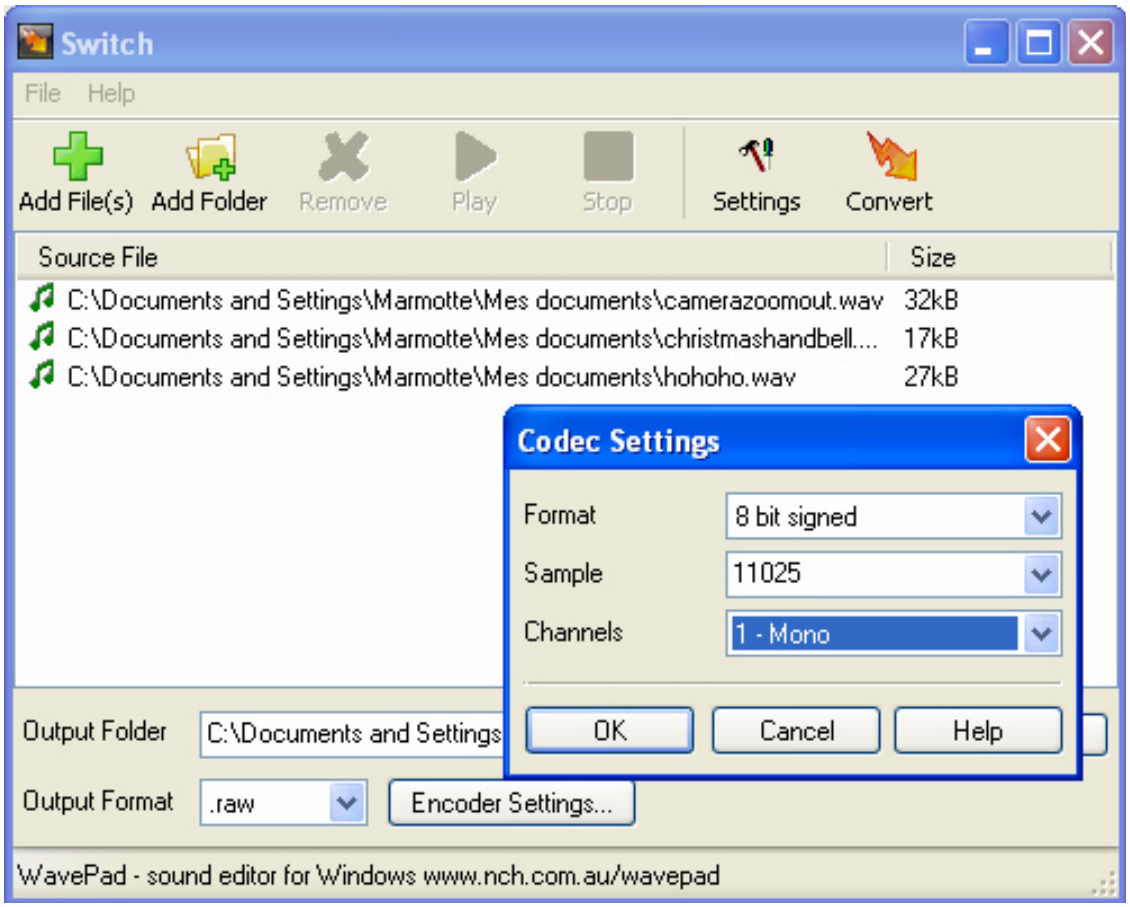


Figura B.8: Ejemplo exportar a .raw. Fuente: Palib.info

Para que la conversión sea válida es necesario dejar el formato de 8 bit signed, sin embargo las otras dos opciones se pueden mejorar para obtener una mejor calidad. Por ejemplo, poniendo mayor cantidad de muestreo (sample) o incluso establecer sonido estéreo.

Para poder reproducir un archivo de sonido raw en una aplicación es necesario que este se aloje en la carpeta data que está al mismo nivel que la carpeta source, en caso de que no exista será necesario crearla. El código siguiente reproduce un sonido al pulsar el botón *A*:

```
// Ejemplo de sonido. Formato Raw
// Includes
#include <PA9.h> // Include para PA_Lib
#include "saberoff.h// Include para el sonido
//(se aloja en la carpeta data en formato .raw)
// Función: main()
int main(int argc, char ** argv){
```

```
    PA_Init(); // Inicializa PA_Lib
    PA_InitVBL(); // Inicializa un standard VBL
    PA_InitText(0, 0);
    PA_InitSound(); // Inicializa el sistema de sonido
    PA_OutputSimpleText(0, 0, 10, "Pulsa A para reproduc. el sonido");
    // Bucle infinito
    while (1){ // Reproduce el sonido si A ha sido pulsado
        if (Pad.Newpress.A)
            PA_PlaySimpleSound(0, saberoff);
        PA_WaitForVBL();
    }
    return 0;
} // Fin de main()
```

Lo primero importante que se hace antes de comenzar las instrucciones es incluir el archivo raw con #include "saberoff.h". Después cuando se quiera reproducir el sonido bastará con usar su nombre ya que se ha establecido como un puntero al contenido de dicho fichero. A continuación es necesario inicializar el sistema de sonido de PAlib[6] (sirve también para la reproducción de archivos mod) estableciendo el tipo de archivo por defecto a muestreo *11025* y *8 bit* signed format. Por último queda efectuar la acción de reproducir el sonido en el momento que sea necesario, en nuestro caso cuando se pulse el botón *A*. Para este fin se llama a la función **PA_PlaySimpleSound(PA_Channel, sonido)** a la que se le indica un canal de salida de indice cero al siete y un archivo de sonido previamente incluido en el código. Pueden usarse hasta ocho canales de reproducción al mismo tiempo. Por defecto los otros ocho canales disponibles están reservados para la reproducción de archivos mod, aunque esto es modificable.

Archivos de sonido mod

Los archivos mod tienen como ventaja que ocupan muy poco espacio y suenan bien. Sin embargo, no son fáciles de crear y por tanto se recomienda buscar en la web los archivos ya existentes en bancos de sonidos gratuitos como por ejemplo: *The Mod Archive*, *Exótica*, and *ModLand*. Ahora mediante un ejemplo se verá como reproducir estos archivos:

```
// Reproduce un archivo mod
// Includes
#include <PA9.h> // Include para PA_Lib
#include "modfile.h// Include el archivo mod (el archivo .mod se aloja
// en la carpeta data)
// Función: main()
int main(int argc, char ** argv){
    PA_Init();
```

```
    PA_InitVBL();
    PA_InitSound(); // Inicializa el sonido
    PA_InitText(0, 0); PA_InitText(1, 0);
    PA_PlayMod(modfile); // Reproduce el archivo mod
    PA_OutputText(0, 5, 10, Reproduciendo mod...");
    u8 i;
    while(1){ // Bucle infinito
        // Muestra si los canales están ocupados
        for (i = 0; i < 16; i++)
            PA_OutputText(1, 0, i, Çanal %02d ocupado : %d ", i, PA_SoundChannelIsBusy(i));
// 0 libre, 1 ocupado
        PA_WaitForVBL();
    }
    return 0;
} // Fin de main()
```

En este caso se llama a **PA_PlayMod(mod_snd)**. Una cosa que hay que comentar es que el archivo mod puede tener hasta dieciséis canales, pero es recomendable usar ocho para dejar los ocho restantes libres para efectos de sonido.

Funciones adicionales de sonido

Hasta ahora se han visto las funciones básicas para reproducir los dos tipos de formatos que permite PAlib[6], sin embargo existen más funcionalidades a parte de reproducir sonidos. Las más importantes son las siguientes.

Para archivos mod existe la posibilidad de detener la reproducción con la función **PA_StopMod()** y también es posible pausarlo con **PA_PauseMod(u8)**. Una función útil para archivos raw es la siguiente: **PA_SetDefaultSound (u8 volume, int freq, s16 format)**, que establece las opciones del archivo como son volumen, frecuencia y formato.

Además el volumen se puede modificar a nivel de canal con **PA_SetSoundChannelVol (u8 PA_Channel, u8 Volume)** y también es posible controlar el master del volumen con **PA_SetSoundVol (u8 Volume)**.

B.4.9. Programación Hardware DS

En este apartado se estudia las diferentes funciones que aporta PAlib[6] para trabajar con el hardware de DS como por ejemplo apagar la consola, información del usuario, reloj...

Fecha y hora

Para poder gestionar en las aplicaciones diferentes eventualidades a lo largo del tiempo, será necesario conocer las siguientes variables que harán uso del reloj interno que trae la consola y que debe estar previamente configurado por el usuario de la videoconsola. Con este sencillo ejemplo se accede a todo el contenido necesario:

```
// Muestra la fecha y hora
#include <PA9.h>
// Función Main
int main(void){
    // Inicialización de PAlib
    PA_Init();
    PA_InitVBL();
    // Inicializa el texto
    PA_InitText(1, 0);
    while(1){
        // Día, Mes y Año
        PA_OutputText(1, 2, 10, " %02d/ %02d/ %02d", PA_RTC.Day, PA_RTC.Month,
PA_RTC.Year);
        // Hora HH:MM SS
        PA_OutputText(1, 2, 12, " %02d: %02d %02d segundos", PA_RTC.Hour,
        PA_RTC.Minutes, PA_RTC.Seconds);
        PA_WaitForVBL();
    }
    return 0;
}
```

PA_RTC es una estructura que se actualiza cada frame la cual contiene toda la información sobre la fecha y hora actual. Estas son las variables que contiene:

- "**PA_RTC.Day**", de 1 a 31.
- "**PA_RTC.Month**", de 1 a 12.
- "**PA_RTC.Year**", de 00 (para 2000) a 99 (para 2099).
- "**PA_RTC.Hour**", de 0 a 23.
- "**PA_RTC.Minutes**", de 0 a 59.
- "**PA_RTC.Seconds**", de 0 a 59.

Información de usuario

La información de usuario se encuentra almacenada en la variable **PA_UserInfo**. A continuación un ejemplo que la muestra por pantalla:

```
// Muestra la información del usuario
#include <PA9.h>
// Función Main
int main(void){
    // Inicialización de PAlib
    PA_Init();
    PA_InitVBL();
    // Inicializa el texto
    PA_InitText(1, 0);
    while(1){
        // Nombre de usuario, cumpleaños
        PA_OutputText(1, 2, 10, "Nombre usuario : %s, %02d/ %02d",
        PA_UserInfo.Name, PA_UserInfo.BdayDay,
        PA_UserInfo.BdayMonth);
        // Alarma
        PA_OutputText(1, 2, 12, .^Alarma : %02d: %02d", PA_UserInfo.AlarmHour, PA_UserInfo.AlarmMinute);
        // Idioma DS (0 - japonés, 1 - inglés,
        // 2 - francés .. 5 - español)
        PA_OutputText(1, 2, 14, "Idioma : %d", PA_UserInfo.Language);
        // Mensaje especial
        PA_OutputText(1, 2, 16, "Mensaje : %s", PA_UserInfo.Message);
        PA_WaitForVBL();
    }
    return 0;
}
```

El contenido de la estructura **PA_UserInfo** es el siguiente:

- **PA_UserInfo.Name**, nombre de usuario.
- **PA_UserInfo.BdayDay**, día de cumpleaños.
- **PA_UserInfo.BdayMonth**, mes de cumpleaños.
- **PA_UserInfo.Language**, idioma identificado por un número (*0 - japonés, 1 - inglés, 2 – francés, 3 – Alemán, 4 – Italiano, 5 - español*).

- **PA_UserInfo.Message**, mensaje de bienvenida establecido por el propietario.
- **PA_UserInfo.AlarmHour**, hora de la alarma (*de 0 a 23*).
- **PA_UserInfo.AlarmMinute**, minuto de la alarma.
- **PA_UserInfo.Color**, color elegido por el propietario.

Pausa al cerrar

En los juegos comerciales de se puede observar como al plegar la videoconsola cerrando las tapas, el juego se pausa automáticamente hasta que se vuelve a abrir. Esto se puede realizar también usando PAlib[6] llamando simplemente a la función **PA_CheckLid (void)** que en caso de detectar que se han cerrado las tapas, pausa la consola y devuelve un el valor uno. Aquí se puede ver un ejemplo:

```
// Muestra la información del usuario
#include <PA9.h>
// Función Main
int main(void){
    // Inicialización de PAlib
    PA_Init();
    PA_InitVBL();
    // Establece un fondo de color rojo para poder comprobar
    // si está pausada o no fácilmente
    PA_SetBgPalCol(0, 0, PA_RGB(31, 0, 0));
    while (1){
        PA_CheckLid(); // Comprueba si está cerrada
        PA_WaitForVBL();
    }
    return 0;
}
```

Para probar este ejemplo es necesario disponer de una consola DS.

Iluminación de pantallas

Con la función void **PA_SetScreenLight (u8 screen, u8 light)** se puede activar/desactivar cualquiera de las dos pantallas.

B.4.10. Programación 3D

Hasta ahora se ha visto la mayor parte de herramientas que dispone un desarrollador para construir aplicaciones en Nintendo DS. La parte que viene a continuación trata sobre las posibilidades que ofrecen las librerías ya comentadas para trabajar con escenarios en 3D. Se verá cómo poder inicializar estas funciones, pintar elementos básicos como quads y triángulos, aplicar transformaciones 3D, manejar texturas...

El soporte para programación gráfica 3D, lo ofrece una implementación de openGL dentro de la librería libnds[2]. PAlib[6] también incluye un módulo de funciones 3D que facilitan las tareas de libnds[2], aunque se centra en el tratamiento de sprites 3D. Por tanto, y como el objetivo final de este proyecto consistirá en mostrar un escenario 3D no será tan necesario hacer uso del módulo 3D incluido dentro de PAlib[6].

Inicializar funciones 3D

Hasta ahora la librería PAlib[6] que se ha estudiado era la responsable de llamar a las funciones de más bajo nivel que componen la otra librería libnds[2], simplificando significativamente la tarea. Sin embargo, como se ha explicado anteriormente, el módulo de PAlib[6] para la programación gráfica 3D no llega a sustituir completamente a la librería libnds[2]. Por tanto, a partir de este momento será necesario combinar ambas librerías para poder simplificar las tareas lo máximo posible. El siguiente ejemplo no muestra nada por la pantalla, simplemente inicializa todo lo necesario para que después se puedan pintar componentes 3D correctamente.

```
// Ejemplo inicializar 3D
#include <PA9.h>
//Función encargada de pintar la escena
int DrawGLScene();
// Función Main
int main(void){
    // Inicialización de PAlib
    PA_Init();
    PA_InitVBL();
    //Inicialización 3D con PAlib
    PA_Init3D();
    // Habilita antialias
    glEnable(GL_ANTIALIAS);
    // Activa matriz de proyección para modificar
    glMatrixMode(GL_PROJECTION);
    // Carga la matriz identidad
    glLoadIdentity();
```

```
        // Pone la proyección en perspectiva ajustando
        // al tamaño de pantalla de NDS
        gluPerspective(35, 256.0 / 192.0, 0.1, 100);
        // Orienta la cámara
        gluLookAt( 0.0, 0.0, 1.0, // Posición de la cámara
            0.0, 0.0, 0.0, // Hacia donde apunta
            0.0, 1.0, 0.0); // Vector up
        while (1){
            // Push de la matriz activa (salva el estado)
            glPushMatrix();
            DrawGLScene();
            // Pop de la matriz actual (carga el estado salvado en la pila)
            glPopMatrix(1);
            PA_WaitForVBL();
            // Limpia el buffer de la pantalla
            glFlush(0);
        }
        return 0;
}
int DrawGLScene(void){
        // Carga la matriz identidad sobre la matriz activa
        glLoadIdentity();
        return TRUE;
}
```

Comienza incluyendo la librería PAlib[6] como siempre se ha hecho, ya que se seguirán usando algunas funciones de ésta. No es necesario incluir la librería libnds[2] porque ya se hace de manera implícita al cargar PAlib[6]. Después se declara una función llamada **DrawGLScene**(), que contendrá en los ejemplos siguientes la escena 3D a pintar. Dentro de la función principal main se inicializa PAlib[6] como siempre. A continuación se inicializan las funciones 3D llamando a **PA_Init3D()**. En su interior se inicializa lo básico para poder usar las funciones 3D pero será necesario establecer más adelante una serie de características de configuración que serán distintas según las circunstancias. La función **glEnable** se encarga de habilitar propiedades, en este caso activa un filtro antialias que mejora la visualización, por ejemplo suavizando los contornos de las figuras. Con la función **glMatrixMode** se establece a cuál de las matrices que utiliza openGL se le aplicarán las transformaciones resultantes de las funciones que aparezcan a continuación de dicha llamada, es decir, que mientras no se establezca una matriz con **glMatrixMode**, esta no podrá ser modificada. Las matrices más usadas de openGL son dos: *GL_PROJECTION* y *GL_MODELVIEW*. La primera es la responsable de establecer la perspectiva de nuestra escena. La segunda matriz contiene las transformaciones aplicadas a los objetos de la escena. La siguiente instrucción es **glLoadIdentity()**, cuya tarea es cargar la matriz identidad

sobre la matriz establecida con glMatrixMode. De este modo se inicializa el contenido de la matriz eliminando todas las transformaciones que se hubieran aplicado a esta. Estando activada la matriz de proyección, es necesario cargar sobre esta el tipo de proyección que se quiere mostrar en la escena. Para ello se llama a la función **gluPerspective(35, 256.0/192.0, 0.1, 100);** cuyos parámetros significan respectivamente: treinta y cinco grados de ángulo de visión de altura (eje Y), relación de aspecto acho/alto, *256/192*, que determina el ángulo de visión a lo largo del eje X, distancia desde el observador al plano de corte cercano, distancia desde el observador al plano de corte lejano. Lo siguiente es posicionar y orientar la cámara correctamente con **gluLookAt**. Sus parámetros se dividen en tres ternas: la primera establece la coordenada donde estará situada la cámara, la segunda el punto del escenario 3D hacia el que apunta la cámara, la última terna expresa el vector alzado de la cámara que forma 90^o con el vector que se construye entre la posición hacia donde mira y la posición donde se encuentra. Dentro del bucle principal, se llama a **glPushMatrix** que guarda la matriz activada en lo alto de una pila. Con esto se salva su estado para poder recuperarlo más adelante desde la pila. Después se llama a la función que pintará la escena. Finalmente se recupera el estado de la matriz de transformaciones de modelos con **glPopMatrix** y se limpia el buffer de memoria con **glFlush**.

Polígonos

Cuando ya se ha inicializado convenientemente la librería openGL, el sistema está dispuesto para dibujar elementos 3D y poder mostrarlos. Por tanto, sólo será necesario modificar la función **DrawGLScene()** ya que es la que se encarga en cada iteración del bucle infinito de pintar sobre la pantalla los elementos deseados:

```
// Aquí es donde se pinta todo
int DrawGLScene(){
    // Activa la matriz de modelos de escena
    glMatrixMode(GL_MODELVIEW);
    // Inicializa la matriz activa
    glLoadIdentity();
    // Mueve 1.5 unidades a la izquierda y 6.0 hacia la pantalla
    glTranslatef(-1.5f,0.0f,-3.0f);
    // Formato de polígonos, establece alpha a 31 (desactivado transparencias) y el modo
de procesado de
    // superficies ocultas a none
    glPolyFmt(POLY_ALPHA(31) | POLY_CULL_NONE);
    // Dibuja triángulos
    glBegin(GL_TRIANGLES);
    glColor3f(1.0f,0.0f,0.0f); // Establece el color de vértice a Rojo
    glVertex3f( 0.0f, 1.0f, 0.0f); // Vértice superior
    glColor3f(0.0f,1.0f,0.0f); // Establece el color de vértice a Verde
    glVertex3f(-1.0f,-1.0f, 0.0f); // Vértice inferior izquierdo
```

```
        glColor3f(0.0f,0.0f,1.0f); // Establece el color de vértice a Azul
        glVertex3f( 1.0f,-1.0f, 0.0f); // Vértice inferior derecho
        // Fin Triángulo
        glEnd();
        // Desplaza 3 unidades a la derecha
        glTranslatef(3.0f,0.0f,0.0f);
        // Establece el color para los siguientes vértices a azul
        glColor3f(0.5f,0.5f,1.0f);
        // Dibuja un Quad
        glBegin(GL_QUADS);
        glVertex3f(-1.0f, 1.0f, 0.0f); // Vértice superior izquierdo
        glVertex3f( 1.0f, 1.0f, 0.0f); // Vértice superior derecho
        glVertex3f( 1.0f,-1.0f, 0.0f); // Vértice inferior derecho
        glVertex3f(-1.0f,-1.0f, 0.0f); // Vértice inferior izquierdo
        // Fin Quad
        glEnd();
        return TRUE;
}
```

OpenGL trabaja cargando sobre sus matrices las transformaciones que se van solicitando. Cada vez que se realiza una nueva operación, ya sea pintar un polígono, rotar la escena o cualquier otra tarea, esta se aplica sobre el resto de modificaciones anteriores. Por este motivo lo que se realiza al comienzo de la función es establecer como matriz activa la matriz que carga los modelos de la escena junto con sus transformaciones y reiniciarla, es decir, se carga la matriz identidad sobre ella perdiendo su contenido anterior. Así queda lista para trabajar con ella desde el principio. La siguiente instrucción se corresponde con **void glTranslatef(float x, float y, float z)** que como su nombre indica realiza una translación sobre los elementos que se pinten a continuación de esta instrucción.

Si no se pusiera esta instrucción, lo que se pintara se haría sobre el punto (0, 0) que se encuentra en el centro de la pantalla. Los parámetros que se le pasan a esta función le indican la cantidad de desplazamiento en x (horizontal), y (vertical) y z (profundidad) respectivamente. Siendo las coordenadas positivas las que desplazan hacia la derecha, arriba y hacia la pantalla en cada caso. Para este ejemplo, la translación es en *1.5 unidades* hacia la izquierda y seis unidades alejadas de la pantalla con respecto del usuario.

Con la función **glPolyFmt** se establecen las propiedades asociadas a los polígonos de la escena. Para este caso se deja el valor de alpha (transparencia) al valor treinta y uno, sin transparencia, y se establece que ambos lados de los polígonos sean renderizados.

Después comienza el pintado de un triángulo con **glBegin(GL_TRIANGLES)**. Desde esta línea hasta **glEnd()** todos los pares de tres vértices que se pinten formarán triángulos. Existen varios tipos de figuras simples que se pueden pintar. Para esta implementación de openGL son las siguientes: *GL_TRIANGLES*, *GL_QUADS*, *GL_TRIANGLE_STRIP* y *GL_QUAD_STRIP*. La primera es

para dibujar triángulos, la segunda forma un cuadrado por cada grupo de cuatro vértices, y las dos últimas son similares a las dos primeras salvo que cada grupo de vértice se encadenará al siguiente. En este ejemplo, dentro de los bloques **glBegin** y **glEnd** aparecen dos funciones distintas. La más significativa es **void glVertex3f(float x, float y, float z)** que pinta un vértice de la figura en las coordenadas x, y, z indicadas. La otra función es **void glColor3f(float r, float g, float b)** que establece el color a los vértices que se pinten a continuación según la terna rojo, verde y azul. En el ejemplo a cada vértice del triángulo se le aplica un color diferente, mientras que el cuadrado está pintado con todos los vértices del mismo color azul.

Rotación

En este caso además de modificar la función **DrawGLScene**, también se han creado dos variables globales para controlar las rotaciones. Estas guardarán un valor que se modificará tras cada iteración, para que dé la sensación de que las figuras están girando cuando en realidad se están volviendo a pintar con distintas inclinaciones.

```
float rtri; // Ángulo para el triángulo
float rquad; // Ángulo para el cuadrado
int DrawGLScene(){ // Aquí es donde se pinta todo
    // Activa la matriz de modelos de escena
    glMatrixMode(GL_MODELVIEW);
    // Inicializa la matriz activa
    glLoadIdentity();
    // Mueve 1.5 unidades a la izquierda y 6.0 hacia la pantalla
    glTranslatef(-1.5f,0.0f,-4.0f);
    // Gira el triángulo en el eje Y
    glRotatef(rtri,0.0f,1.0f,0.0f);
    // Formato de polígonos, establece alpha a 31 (desactivado transparencias) y el modo
de procesado de
    // superficies ocultas a none
    glPolyFmt(POLY_ALPHA(31) | POLY_CULL_NONE);
    // Dibuja triángulos
    glBegin(GL_TRIANGLES);
    glColor3f(1.0f,0.0f,0.0f); // Establece el color de vértice a Rojo
    glVertex3f( 0.0f, 1.0f, 0.0f); // Vértice superior
    glColor3f(0.0f,1.0f,0.0f); // Establece el color de vértice a Verde
    glVertex3f(-1.0f,-1.0f, 0.0f); // Vértice inferior izquierdo
    glColor3f(0.0f,0.0f,1.0f); // Establece el color de vértice a Azul
    glVertex3f( 1.0f,-1.0f, 0.0f); // Vértice inferior derecho
    // Fin Triángulo
```

```
        glEnd();
        // Reinicia la actual matriz Modelview
        glLoadIdentity();
        // Mueve 1.5 unidades a la derecha y 6.0 hacia la pantalla
        glTranslatef(1.5f,0.0f,-4.0f);
        // Gira el cuadrado en el eje X
        glRotatef(rquad,1.0f,0.0f,0.0f);
        // Establece el color para los siguientes vértices a azul
        glColor3f(0.5f,0.5f,1.0f);
        // Dibuja un Quad
        glBegin(GL_QUADS)
        glVertex3f(-1.0f, 1.0f, 0.0f); // Vértice superior izquierdo
        glVertex3f( 1.0f, 1.0f, 0.0f); // Vértice superior derecho
        glVertex3f( 1.0f,-1.0f, 0.0f); // Vértice inferior derecho
        glVertex3f(-1.0f,-1.0f, 0.0f); // Vértice inferior izquierdo
        // Fin Quad
        glEnd();
        // Aumenta la rotación variable para el triángulo
        rtri+=0.9f;
        // Decrementa la rotación variable para el triángulo
        rquad-=0.75f;
        return TRUE;
    }
```

La función que aparece nueva esta ocasión es void **glRotatef(float angle, float x, float y, float z)**. Se encarga de transformar la escena con una rotación a todos los elementos que se pinten a continuación, Su primer parámetro es el ángulo de rotación y los tres siguientes son los ejes sobre los que se aplica. En esta aplicación se realiza una rotación diferente a cada polígono. Al comienzo de la función se inicializa la matriz olvidando transformaciones previas y se aplica una traslación como sucedía en el ejemplo anterior. Después se aplica la rotación sobre el eje Y. Se pinta el triángulo. Lo siguiente es Inicializar de nuevo la matriz modelview ya que el objetivo de este ejemplo es aplicar dos rotaciones diferentes e independientes entre sí a cada polígono.

La primera rotación sobre el eje Y, y la segunda sobre el eje X. Si se aplicara la translación y la rotación sin más sucedería que se acumularían ambas rotaciones, ya que sólo existe una matriz de escena modelview que es común a todos, por tanto es necesario volverla a inicializar para que las aplicaciones previas no se acumulen con las siguientes. Por tanto, a partir de su nueva inicialización será necesario volver a trasladar con respecto a la coordenada (0, 0) y aplicar la rotación sobre el objeto. Es interesante comentar la línea de la segunda inicialización de la matriz para ver qué sucede en tal caso. Por último se actualizan los valores de las variables globales para conseguir el efecto de girar.

Figuras Básicas 3D

Este apartado no añade ninguna función nueva, sin embargo, resulta necesario para comprender el nivel de abstracción con el que trabaja la librería de programación gráfica openGL.

Para poder dibujar objetos 3D es necesario representar las caras que lo forman, vértice por vértice. Esto quiere decir que, para construir una pirámide cuadrangular hace falta pintar cuatro caras, prescindiendo de la cara inferior que serviría de base cerrando la figura. Si la figura a representar es un cubo serán necesarias las seis caras que forman un dado. Para la representación de cada cara, se usa el tipo de estructura más conveniente de las vistas anteriormente. Recordemos que básicamente en este implementación para DS existen dos: *GL_ TRIANGLES* y *GL_ QUADS* de tres y cuatro vértices respectivamente. Las otras dos que existen *GL_ TRIANGLE_ STRIP* y *GL_ QUAD_ TRIP* son agrupaciones de elementos de las otras dos.

A continuación el código del ejemplo que es una extensión del anterior, ya que donde había un triángulo ahora se dibuja una pirámide y donde aparecía un cuadrado ahora hay un cubo. Existe otra modificación, en lugar de girar el cubo sobre un eje ahora lo hace sobre dos.

```
// Aquí es donde se pinta todo
int DrawGLScene(){
    // Activa la matriz de modelos de escena
    glMatrixMode(GL_MODELVIEW);
    // Inicializa la matriz activa
    glLoadIdentity();
    // Mueve 1.5 unidades a la izquierda y 6.0 hacia la pantalla
    glTranslatef(-1.5f,0.0f,-6.0f);
    // Gira el triángulo en el eje Y
    glRotatef(rtri,0.0f,1.0f,0.0f);
    // Formato de polígonos, establece alpha a 31 (desactivado transparencias) y el modo
de procesado de
    // superficies ocultas a none
    glPolyFmt(POLY_ALPHA(31) | POLY_CULL_NONE);
    // Dibuja triángulos
    glBegin(GL_TRIANGLES);
    glColor3f(1.0f,0.0f,0.0f); // Rojo
    glVertex3f( 0.0f, 1.0f, 0.0f); // Vértice superior (Frente)
    glColor3f(0.0f,1.0f,0.0f); // Verde
    glVertex3f(-1.0f,-1.0f, 1.0f); // Vértice inferior izquierdo (Frente)
    glColor3f(0.0f,0.0f,1.0f); // Azul
    glVertex3f( 1.0f,-1.0f, 1.0f); // Vértice inferior derecho (Frente)
    glColor3f(1.0f,0.0f,0.0f); // Rojo
```

```
glVertex3f( 0.0f, 1.0f, 0.0f); // Vértice superior (Derecha)
glColor3f(0.0f,0.0f,1.0f); // Azul
glVertex3f( 1.0f,-1.0f, 1.0f); // Vértice inferior izquierdo (Derecha)
glColor3f(0.0f,1.0f,0.0f); // Verde
glVertex3f( 1.0f,-1.0f, -1.0f); // Vértice inferior derecho (Derecha)
glColor3f(1.0f,0.0f,0.0f); // Rojo
glVertex3f( 0.0f, 1.0f, 0.0f); // Vértice superior (Atrás)
glColor3f(0.0f,1.0f,0.0f); // Verde
glVertex3f( 1.0f,-1.0f, -1.0f); // Vértice inferior izquierdo (Atrás)
glColor3f(0.0f,0.0f,1.0f); // Azul
glVertex3f(-1.0f,-1.0f, -1.0f); // Vértice inferior derecho (Atrás)
glColor3f(1.0f,0.0f,0.0f); // Rojo
glVertex3f( 0.0f, 1.0f, 0.0f); // Vértice superior (Izquierda)
glColor3f(0.0f,0.0f,1.0f); // Azul
glVertex3f(-1.0f,-1.0f,-1.0f); // Vértice inferior izquierdo (Izquierda)
glColor3f(0.0f,1.0f,0.0f); // Verde
glVertex3f(-1.0f,-1.0f, 1.0f); // Vértice inferior derecho (Izquierda)
// Fin Triángulo
glEnd();
// Reinicia la actual matriz Modelview
glLoadIdentity();
// Mueve 1.5 unidades a la derecha y 6.0 hacia la pantalla
glTranslatef(1.5f,0.0f,-6.0f);
// Gira el cuadrado en el eje X
glRotatef(rquad,1.0f,1.0f,1.0f);
// Dibuja un Quad
glBegin(GL_QUADS);
glColor3f(0.0f,1.0f,0.0f); // Verde
glVertex3f( 1.0f, 1.0f,-1.0f); // Vértice superior derecho (Arriba)
glVertex3f(-1.0f, 1.0f,-1.0f); // Vértice superior izquierdo (Arriba)
glVertex3f(-1.0f, 1.0f, 1.0f); // Vértice inferior izquierdo (Arriba)
glVertex3f( 1.0f, 1.0f, 1.0f); // Vértice inferior derecho (Arriba)
glColor3f(1.0f,0.5f,0.0f); // Color naranja
glVertex3f( 1.0f,-1.0f, 1.0f); // Vértice superior derecho (Abajo)
glVertex3f(-1.0f,-1.0f, 1.0f); // Vértice superior izquierdo (Abajo)
glVertex3f(-1.0f,-1.0f,-1.0f); // Vértice inferior izquierdo (Abajo)
glVertex3f( 1.0f,-1.0f,-1.0f); // Vértice inferior derecho (Abajo)
```

```
        glColor3f(1.0f,0.0f,0.0f); // Rojo
        glVertex3f( 1.0f, 1.0f, 1.0f); // Vértice superior derecho (Frente)
        glVertex3f(-1.0f, 1.0f, 1.0f); // Vértice superior izquierdo (Frente)
        glVertex3f(-1.0f,-1.0f, 1.0f); // Vértice inferior izquierdo (Frente)
        glVertex3f( 1.0f,-1.0f, 1.0f); // Vértice inferior derecho (Frente)
        glColor3f(1.0f,1.0f,0.0f); // Amarillo
        glVertex3f( 1.0f,-1.0f,-1.0f); // Vértice superior derecho (Atrás)
        glVertex3f(-1.0f,-1.0f,-1.0f); // Vértice superior izquierdo (Atrás)
        glVertex3f(-1.0f, 1.0f,-1.0f); // Vértice inferior izquierdo (Atrás)
        glVertex3f( 1.0f, 1.0f,-1.0f); // Vértice inferior derecho (Atrás)
        glColor3f(0.0f,0.0f,1.0f); // Azul
        glVertex3f(-1.0f, 1.0f, 1.0f); // Vértice superior derecho (Izquierda)
        glVertex3f(-1.0f, 1.0f,-1.0f); // Vértice superior izquierdo (Izquierda)
        glVertex3f(-1.0f,-1.0f,-1.0f); // Vértice inferior izquierdo (Izquierda)
        glVertex3f(-1.0f,-1.0f, 1.0f); // Vértice inferior derecho (Izquierda)
        glColor3f(1.0f,0.0f,1.0f); // Violeta
        glVertex3f( 1.0f, 1.0f,-1.0f); // Vértice superior derecho (Derecha)
        glVertex3f( 1.0f, 1.0f, 1.0f); // Vértice superior izquierdo (Derecha)
        glVertex3f( 1.0f,-1.0f, 1.0f); // Vértice inferior izquierdo (Derecha)
        glVertex3f( 1.0f,-1.0f,-1.0f); // Vértice inferior derecho (Derecha)
        // Fin Quad
        glEnd();
        // Aumenta la rotación variable para el triángulo
        rtri+=0.9f;
        // Decrementa la rotación variable para el triángulo
        rquad-=0.75f;
        return TRUE;
    }
```

Texturas

Ya se han visto las funcionalidades básicas que aporta esta implementación de openGL, con las cuales técnicamente se podría hacer cualquier figura y aplicarle cualquier color. Sin embargo, a la hora de mostrarlo dejaría bastante que desear ya que estéticamente no impresionarían demasiado esas formas poligonales con colores tan planos, dando la sensación de algo tosco y no llamaría la atención lo cual suele ser el gran objetivo de cualquier producto audiovisual. Por tanto, quedarían dos aspectos visuales más que mencionar. Uno sería cómo cargar en nuestras aplicaciones modelos 3D realizados con software de diseño especializado y otro cómo se pueden aplicar texturas. En este apartado se verá lo segundo.

Para comenzar una textura es un método para añadir detalle a los gráficos o modelos 3D. La técnica más común es aplicar una imagen ya existente sobre una malla de vértices. Los resultados pueden ser tan buenos como lo sea la textura, pudiendo conseguir que un modelo 3D tenga un aspecto más real. También significa una mejora del rendimiento ya que, por ejemplo, en lugar de modelar una piedra que podría tener una gran cantidad de vértices, debido a su superficie irregular, que harían enlentecer el juego, podría simularse el aspecto de piedra aplicando una textura a un modelo mucho más simple consiguiente la apariencia deseada sin apenas costes de rendimiento. En el siguiente ejemplo se pintará un cubo girando con una textura de caja de madera.

Figura B.9: Textura para el cubo. Fuente: Palib.info

```
// Texturas
#include <PA9.h>
#include Çrate.h// Textura a cargar
float rtri;
float rquad;
int textureID; // variable id de la textura
int DrawGLScene();
int main(void){
    PA_Init();
    PA_InitVBL();
    PA_Init3D();
    glEnable(GL_ANTIALIAS);
    glEnable(GL_TEXTURE_2D); // Habilita el uso de texturas
    glGenTextures(1, &textureID); //Genera una textura sobre textureID
    glBindTexture(0, textureID); // Vincula la textura
    glTexImage2D( // Asigna sobre la textura 0 la textura Crate
        0, 0, GL_RGB, TEXTURE_SIZE_128 , TEXTURE_SIZE_128,
        0, TEXGEN_TEXCOORD, (u8*)Crate);
```

```
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(35, 256.0 / 192.0, 0.1, 100);
    gluLookAt( 0.0, 0.0, 1.0,
        0.0, 0.0, 0.0,
        0.0, 1.0, 0.0);
    while (1){
        glPushMatrix();
        DrawGLScene();
        glPopMatrix(1);
        glFlush(0);
        PA_WaitForVBL();
    }
    return 0;
}
int DrawGLScene(){
    // Inicializa la matriz de texturas
    glMatrixMode(GL_TEXTURE);
    glLoadIdentity();
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    // Establece las propiedades de los materiales
    glMaterialf(GL_AMBIENT, RGB15(31,31,31));
    glMaterialf(GL_DIFFUSE, RGB15(31,31,31));
    glMaterialf(GL_SPECULAR, BIT(15) | RGB15(31,31,31));
    glMaterialf(GL_EMISSION, RGB15(31,31,31));
    glTranslatef(0.0f,0.0f,-4.0f);
    glRotatef(rtri,0.6f,1.0f,0.8f);
    glPolyFmt(POLY_ALPHA(31) | POLY_CULL_NONE);
    glBindTexture(0, textureID); // Carga la textura 0
    glBegin(GL_QUADS);
    // Cara frontal
    glNormal3f( 0.0f, 0.0f, 1.0f);
    glTexCoord2f(0.0f, 0.0f); glVertex3f(-1.0f, -1.0f, 1.0f);
    glTexCoord2f(1.0f, 0.0f); glVertex3f( 1.0f, -1.0f, 1.0f);
    glTexCoord2f(1.0f, 1.0f); glVertex3f( 1.0f, 1.0f, 1.0f);
    glTexCoord2f(0.0f, 1.0f); glVertex3f(-1.0f, 1.0f, 1.0f);
```

```
    // Cara trasera
    glNormal3f( 0.0f, 0.0f,-1.0f);
    glTexCoord2f(1.0f, 0.0f); glVertex3f(-1.0f, -1.0f, -1.0f);
    glTexCoord2f(1.0f, 1.0f); glVertex3f(-1.0f, 1.0f, -1.0f);
    glTexCoord2f(0.0f, 1.0f); glVertex3f( 1.0f, 1.0f, -1.0f);
    glTexCoord2f(0.0f, 0.0f); glVertex3f( 1.0f, -1.0f, -1.0f);
    // Cara superior
    glNormal3f( 0.0f, 1.0f, 0.0f);
    glTexCoord2f(0.0f, 1.0f); glVertex3f(-1.0f, 1.0f, -1.0f);
    glTexCoord2f(0.0f, 0.0f); glVertex3f(-1.0f, 1.0f, 1.0f);
    glTexCoord2f(1.0f, 0.0f); glVertex3f( 1.0f, 1.0f, 1.0f);
    glTexCoord2f(1.0f, 1.0f); glVertex3f( 1.0f, 1.0f, -1.0f);
    // Cara inferior
    glNormal3f( 0.0f,-1.0f, 0.0f);
    glTexCoord2f(1.0f, 1.0f); glVertex3f(-1.0f, -1.0f, -1.0f);
    glTexCoord2f(0.0f, 1.0f); glVertex3f( 1.0f, -1.0f, -1.0f);
    glTexCoord2f(0.0f, 0.0f); glVertex3f( 1.0f, -1.0f, 1.0f);
    glTexCoord2f(1.0f, 0.0f); glVertex3f(-1.0f, -1.0f, 1.0f);
    // Cara derecha
    glNormal3f( 1.0f, 0.0f, 0.0f);
    glTexCoord2f(1.0f, 0.0f); glVertex3f( 1.0f, -1.0f, -1.0f);
    glTexCoord2f(1.0f, 1.0f); glVertex3f( 1.0f, 1.0f, -1.0f);
    glTexCoord2f(0.0f, 1.0f); glVertex3f( 1.0f, 1.0f, 1.0f);
    glTexCoord2f(0.0f, 0.0f); glVertex3f( 1.0f, -1.0f, 1.0f);
    // Cara izquierda
    glNormal3f(-1.0f, 0.0f, 0.0f);
    glTexCoord2f(0.0f, 0.0f); glVertex3f(-1.0f, -1.0f, -1.0f);
    glTexCoord2f(1.0f, 0.0f); glVertex3f(-1.0f, -1.0f, 1.0f);
    glTexCoord2f(1.0f, 1.0f); glVertex3f(-1.0f, 1.0f, 1.0f);
    glTexCoord2f(0.0f, 1.0f); glVertex3f(-1.0f, 1.0f, -1.0f);
    glEnd();
    rtri+=0.9f;
    rquad-=0.75f;
    return TRUE;
}
```

Es necesario incluir la textura en el programa de alguna manera. En nuestro caso partimos de una imagen textura.bmp y es necesario convertirla a un formato inteligible por NDS. Para esto usamos una herramienta homebrew llamada TexConv. Existen otras muchas como PAGfxconverter del creador de PAlib[6], pero con esta es suficiente. Para poder convertir la textura es necesario copiarla en dicho directorio y modificar el fichero build.bat. La línea de ejecución tiene la siguiente estructura: TexConv {textura a convertir} {textura destino}. Para este caso la textura origen se llama textura.bmp y como destino le damos el nombre Crate.bin. Ejecutando el archivo build.bat comprobamos que sale una ventana ms-dos que si todo es correcto mostrará un mensaje como el siguiente:

Loading 'textura.bmp'...

Image size: 128 x 128...

Converting...

Done!

Esto indica el nombre de la textura a convertir, su tamaño en ancho por alto y finalmente que la conversión se ha realizado con éxito. Finalmente copiamos el archivo generado, Crate.bin, en la subcarpeta data de nuestra aplicación.

En el código se hace referencia al fichero "Crate.h" que obviando el cambio de extensión se trata de nuestra textura. Después aparece una variable global textureID. Esta será la referencia que tengamos sobre la textura, sobre ella se generará la imagen y se usará cuando haga falta cargarla. La siguiente línea de interés es **glEnable(GL_TEXTURE_2D)**. Como ya se ha comentado, **glEnable** habilita opciones disponibles, y en este caso posibilita poder hacer uso de texturas. Las siguientes tres líneas se encargar de generar la textura. Con **glGenTextures** se generan nombres de texturas, el primer parámetro indica el número de nombres de texturas a ser generadas y el segundo el lugar donde serán generadas. En este ejemplo generamos una sola textura en la variable *textureID*. La función **glBindTexture** permite crear o generar un nombre de textura. El primer parámetro es el objetivo que en openGL indica el tipo de textura, sin embargo, en esta implementación se ignora este parámetro siendo todas las texturas como *GL_TEXTURE_2D*. El siguiente parámetro es el nombre de la textura a vincular. Cuando se llama a esta función las vinculaciones anteriores dejan de existir y las siguientes operaciones referentes a texturas se aplican sobre la textura vinculada. Para especificar la imagen a la textura vinculada se usa **glTexImage2D**. Los parámetros pasados son: objetivo, nivel (obviados, sólo aparecen por compatibilidad con openGL), tipo, ancho, alto, borde (también obviado), formato, textura. Aquí se asocia la imagen con la variable global.

La siguiente línea de código relacionada con la aparición de texturas se encuentra dentro de la función **DrawGLScene**. Al igual que se hace con la matriz de modelos de la escena, se carga la matriz de texturas *GL_TEXTURE* y se inicializa. El siguiente paso es configurar los materiales, en este caso se aplicará a las propiedades de la textura. Estas propiedades se corresponden con la iluminación ambiental, difusa, especular y de emisión. Como ahora no estamos trabajando con iluminación y nos interesa que se vea lo más claro posible, se le dan los máximos valores para que la luz sea blanca. Si por ejemplo se modificarán estos valores se podrían apreciar distintos tonos de colores. Ahora antes de comenzar a pintar las figuras hay que volver a vincular la textura para que esta se aplique sobre los vértices. Se añaden dos funciones nuevas dentro de **glBegin** y **glEnd**. La primera es **glNormal3f** que se le indica el vector normal con tres float. Esto se usa para que la librería gráfica sepa cómo aplicar la iluminación en función de la posición del foco. La otra función es **glTexCoord2f** y expresa la correspondencia de la textura con los vértices, se expresa con dos componentes ya que la textura

está en dos dimensiones. Este proceso es conceptualmente similar a envolver un objeto con un papel estableciendo puntos en común, de este modo, se hace corresponder una coordenada del papel (2D) con una coordenada del objeto (3D).

Iluminación

Antes se ha mencionado un poco el tema de la iluminación, pero en este capítulo se estudia las posibilidades que ofrece la adaptación de openGL para la videoconsola NDS y como iluminar una escena.

En openGL existen varios tipos de luces, con las cuales, se pueden simular las distintas iluminaciones que pueden observarse en el mundo real. Algunas de estas son iluminación global cuya luz es homogénea y cubre la escena por igual, luz de foco con un cuerpo cónico que tiene un diámetro que aumenta con la distancia, luz puntual que se extiende en todas direcciones a partir de su posición... Sin embargo, la versión de openGL que estamos usando sólo ofrece un tipo de luz llamada paralela. Esta tiene como característica que su posición se sitúa en una paralela de la escena en el infinito. Para crear una luz hay que establecer un número que la identifique, un color RGB de iluminación y una coordenada (x,y,z). La posición se expresa normalizada, entre el velor menos uno y el valor más uno, e indica en cual paralela del infinito se sitúa el foco. Por ejemplo, si la coordenada es *(0, +1 ,0)* la luz estará situada por delante de la escena, se verá que ilumina de frente; si la luz está en (0, -1, 0) estará por detrás. Si la posición X es el valor uno, iluminará por la izquierda y el valor menos uno, lo hará por la derecha. Y así con el resto de coordenadas.

En el ejemplo que se va a mostrar en este apartado se muestra un cubo girado a cuyo alrededor existe una luz que desplaza en torno a él. El foco recorre el perímetro de un cuadrado imaginario situado en los bordes infinitos de la escena, de esta manera se observa como el cubo se iluminar parcialmente por cada cara a medida que se desplaza la luz. Estas son las variables y constantes que intervienen en el ejemplo:

```
// Constantes que definen el estado del movimiento de la luz
#define TRAMO_SUPERIOR 0
#define TRAMO_DERECHO 1
#define TRAMO_INFERIOR 2
#define TRAMO_IZQUIERDO 3
// Variables que establecen las propiedades de la luz para este ejemplo
float xCam = -1.0f, yCam = -1.0f;
int tramo = TRAMO_SUPERIOR;
float inc = 0.2f;
```

Las macros se van a usar para definir los estados de movimiento de la luz, respecto al cuadrado imaginario descrito antes. La posición está contenida en las variables *xCam* y *yCam*, el tramo actual en tramo y el desplazamiento en cada instante por *inc*.

En el bucle principal se encarga de mostrar por pantalla la posición actual de la luz, un mensaje para indicar que si se pulsa el botón Start se desactiva la luz, pinta la escena, y actualiza la posición de la luz en función de su posición actual y del tramo que se encuentre recorriendo:

```
while (1){
    PA_ClearTextBg(1);
    // Muestra por pantalla la posición actual de la luz
    PA_OutputText(1,1,1,"Pos x %f3: ",xCam);
    PA_OutputText(1,1,2,"Pos y %f3: ",yCam);
    glPushMatrix();
    DrawGLScene();
    glPopMatrix(1);
    glFlush(0);
    PA_WaitForVBL();
    // Actualiza la posición de la luz
    switch(tramo){
        case TRAMO_SUPERIOR:
            if(xCam < 1.0f)
                xCam += inc;
            else{
                tramo = TRAMO_DERECHO;
                yCam += inc;
            }
            break;
        case TRAMO_DERECHO:
            if(yCam < 1.0f)
                yCam += inc;
            else{
                tramo = TRAMO_INFERIOR;
                xCam -= inc;
            }
            break;
        case TRAMO_INFERIOR:
            if(xCam > -1.0f)
                xCam -= inc;
            else{
                tramo = TRAMO_IZQUIERDO;
```

```
                yCam -= inc;
            }
            break;
        case TRAMO_IZQUIERDO:
            if(yCam > -1.0f)
                yCam -= inc;
            else{
                tramo = TRAMO_SUPERIOR;
                xCam += inc;
            }
            break;
    }
}
```

Por último, dentro de la función **DrawGLScene** se crea la luz con **glLight**, se activa o no en función de si está pulsado el botón Start y finalmente se dibuja el cubo. Es importante darse cuenta de que, es necesario que se definan las normales del objeto para que se pueda iluminar, sino no se vería nada en la pantalla. Igual de necesario es definir las propiedades de iluminación con **glMaterialf**.

```
int DrawGLScene(){
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    // Crea la luz en la posición indicada por xCam e yCam
    glLight(0, RGB15(31,0,0) , floattov10(xCam), floattov10(yCam), 0);
    glTranslatef(0.0f,0.0f,-4.0f);
    glRotatef(45.0f,1.0f,1.0f,0.0f);
    glMaterialf(GL_AMBIENT, RGB15(2,2,2));
    glMaterialf(GL_DIFFUSE, RGB15(20,0,0));
    glMaterialf(GL_SPECULAR, BIT(15) | RGB15(31,0,0));
    glMaterialf(GL_EMISSION, RGB15(15,0,0));
    glMaterialShinyness();
    // Mientras no esté pulsado Start se activa la luz
    if(!Pad.Held.Start){
        glPolyFmt(
            POLY_ALPHA(31) | POLY_CULL_BACK | POLY_FORMAT_LIGHT0);
    }
    else{
        glPolyFmt(POLY_ALPHA(31) | POLY_CULL_BACK);
```

```
}
glColor3f(1.0f,0.0f,0.0f);
glBegin(GL_QUADS);
// Cara frontal
glNormal3f( 0.0f, 0.0f, 1.0f);
glVertex3f(-1.0f, -1.0f, 1.0f);
glVertex3f( 1.0f, -1.0f, 1.0f);
glVertex3f( 1.0f, 1.0f, 1.0f);
glVertex3f(-1.0f, 1.0f, 1.0f);
// Cara trasera
glNormal3f( 0.0f, 0.0f,-1.0f);
glVertex3f(-1.0f, -1.0f, -1.0f);
glVertex3f(-1.0f, 1.0f, -1.0f);
glVertex3f( 1.0f, 1.0f, -1.0f);
glVertex3f( 1.0f, -1.0f, -1.0f);
// Cara superior
glNormal3f( 0.0f, 1.0f, 0.0f);
glVertex3f(-1.0f, 1.0f, -1.0f);
glVertex3f(-1.0f, 1.0f, 1.0f);
glVertex3f( 1.0f, 1.0f, 1.0f);
glVertex3f( 1.0f, 1.0f, -1.0f);
// Cara inferior
glNormal3f( 0.0f,-1.0f, 0.0f);
glVertex3f(-1.0f, -1.0f, -1.0f);
glVertex3f( 1.0f, -1.0f, -1.0f);
glVertex3f( 1.0f, -1.0f, 1.0f);
glVertex3f(-1.0f, -1.0f, 1.0f);
// Cara derecha
glNormal3f( 1.0f, 0.0f, 0.0f);
glVertex3f( 1.0f, -1.0f, -1.0f);
glVertex3f( 1.0f, 1.0f, -1.0f);
glVertex3f( 1.0f, 1.0f, 1.0f);
glVertex3f( 1.0f, -1.0f, 1.0f);
// Cara izquierda
glNormal3f(-1.0f, 0.0f, 0.0f);
glVertex3f(-1.0f, -1.0f, -1.0f);
glVertex3f(-1.0f, -1.0f, 1.0f);
```

```
        glVertex3f(-1.0f, 1.0f, 1.0f);
        glVertex3f(-1.0f, 1.0f, -1.0f);
        glEnd();
        return TRUE;
    }
```

Seleccionar objetos

No hay duda que una de las características diferenciadoras de esta videoconsola, con respecto a otro hardware es el uso de la pantalla táctil mediante el puntero o Stylus. Como se ha visto es posible usar un registro especial para conocer si se ha pulsado sobre la pantalla y en tal caso es posible saber en qué posición. En el ámbito de 2D es fácil conocer si se ha pulsado sobre un elemento u otro si se conocen las posiciones y dimensiones de los elementos simplemente viendo si la posición del puntero está contenida sobre el área de los elementos.

Pero cuando hablamos de 3D la cosa cambia. Los objetos están inmersos en una escena de tres dimensiones y lo único que conocemos de la pulsación es una coordenada en dos dimensiones. Por tanto, se pierde la correspondencia totalmente. En este apartado se verá un "truco" desarrollado por Gabe Ghearing, colaborador de libnds[2].

Antes de nada definimos el objetivo, queremos pintar una escena donde aparezcan varios elementos 3D por ejemplo tres cubos, uno rojo otro verde y otro azul, y que cuando el usuario pulse sobre alguno de ellos el programa sepa sobre cuál lo ha hecho y lo identifique de alguna manera.

La idea es la siguiente, primero se pinta la escena como ya sabemos hacer, luego lo que hay que hacer es usar la función de openGL **gluPickMatrix**, que restringe la zona de renderizado a un área cuadrada que definamos, pintamos sobre una zona reducida en torno a la pulsación del puntero. Como la variable *GFX_POLYGON_RAM_USAGE* almacena la cantidad de polígonos pintados en pantalla, se compara si ha aumentado el número de polígonos al pintar sobre ese pequeño área, en tal caso significará que había un objeto sobre la zona pulsada, si no varía el número de polígonos entonces no había nada en ese área. Bien, ahora se conoce si había algo allá donde estaba el puntero, falta resolver qué elemento era el que ahí estaba. Para esto lo que hay que hacer es ir comprobando el número de polígonos en pantalla por cada elemento que exista. Sin embargo, no tendremos la solución con el primer elemento que aparezca bajo el puntero, ya que puede ser que hubiera varios superpuestos, por tanto habrá que comprobar además si el elemento que está en esa posición es el más cercano a la cámara. Esto se puede resolver con ayuda de una librería que trae libnds[2] también creada por Gabe Ghearing llamada PosTest. Esta contiene un método **PosTestWresult()** que habiendo establecido previamente lo que llama una posición de prueba, devuelve la distancia entre esta y la cámara. De este modo ya sabremos distinguir entre cuál de los objetos se ha pulsado y cuál está más cerca de la cámara. A continuación se muestra el código del ejemplo:

```
// Selección de figuras
// Includes
#include <PA9.h>
#include <nds/arm9/postest.h> // Librería usada para detectar posiciones
```

```
// Definición de tipos
typedef enum {
    NOTHING,
    RED,
    GREEN,
    BLUE
} Clickable;
// Variables globales
Clickable clicked; // Contiene qué ha sido pulsado
int closeW; // Distancia más cercana a la cámara
int polyCount; // Guarda el número de polígonos dibujados
uint32 rotateX = 0;
uint32 rotateY = 0;
// Guarda la posición pulsada
int touchX, touchY;
// usado por gluPickMatrix()
int viewport[]={0,0,255,191};
// Funciones
int DrawGLScene();
// Ejecutada antes de dibujar un objeto durante el recorte
void startCheck();
// Ejecutado después de dibujar un objeto durante el recorte
void endCheck(Clickable obj);
void drawCube(float x, float y, float z, float r, float g, float b);
int main(){
    PA_Init();
    PA_InitVBL();
    PA_Init3D();
    PA_InitText(1,3);
    // Habilita dibujar el borde, para mostrar el objeto seleccionado
    glEnable(GL_OUTLINE);
    // Establece el color del borde
    glSetOutlineColor(0,RGB15(31,31,31));
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(35, 256.0 / 192.0, 0.1, 100);
    gluLookAt( 0.0, 0.0, 1.0,
```

```
            0.0, 0.0, 0.0,
            0.0, 1.0, 0.0);
        while(1){
            PA_ClearTextBg(1);
            if(clicked==RED)
                PA_OutputText(1,1,1,"Pulsado cubo Rojo");
            else if(clicked==GREEN)
                PA_OutputText(1,1,1,"Pulsado cubo Verde");
            else if(clicked==BLUE)
                PA_OutputText(1,1,1,"Pulsado cubo Azul");
            glPushMatrix();
            DrawGLScene();
            glPopMatrix(1);
            glFlush(0);
            PA_WaitForVBL();
            // Actualiza la rotación si se pulsa algún cursor
            if(Pad.Held.Up)
                rotateX += 3;
            if(Pad.Held.Down)
                rotateX -= 3;
            if(Pad.Held.Left)
                rotateY += 3;
            if(Pad.Held.Right)
                rotateY -= 3;
            // Obtiene la posición pulsada
            touchX = Stylus.X;
            touchY = Stylus.Y;
        }
        return 0;
}
void startCheck() {
        while(PosTestBusy()); // Espera a que la posición de prueba termine
            while(GFX_BUSY); // Espera a que se dibujen todos los
                // polígonos del último objeto
                PosTest_Asynch(0,0,0); // Crea una posición de prueba
                // sobre la actual posición trasladada
```

```
            polyCount = GFX_POLYGON_RAM_USAGE; // Guarda el número de
polígonos
            // actuales
}
void endCheck(Clickable obj) {
    while(GFX_BUSY); // Esepera a que los polígonos sean dibujados¡
        while(PosTestBusy()); // Espera a que la posición de prueba termine
            if(GFX_POLYGON_RAM_USAGE>polyCount){ // Si se ha dibujado al-
gún
            // polígono más..
                if(PosTestWresult()<=closeW) {
                    // Si el objeto actual es el más cercano bajo el cursor..
                    closeW=PosTestWresult();
                    clicked=obj;
                }
            }
}
int DrawGLScene(){
    glViewPort(0,0,255,191); // Establece la vista a la pantalla completa
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(30, 256.0 / 192.0, 0.1, 20);
    glMatrixMode(GL_MODELVIEW);
    glPushMatrix(); // Salva el estado de modelview
    {
        glTranslate3f32(0,0,floattof32(-6));
        glRotateXi(rotateX);
        glRotateYi(rotateY);
        glPushMatrix(); // Salva el estao de modelview antes de la
        // primera pasada
        {
            // Dibuja la escena para mostrarla
            if(clicked==RED) {
                // Establece poly ID para mostrar línea externa
                glPolyFmt(
                    POLY_ALPHA(31) | POLY_CULL_NONE | POLY_ID(1));
            } else {
```

```
            // Establece poly ID para no mostrar la línea
            // externa (del mismo color que el fondo)
            glPolyFmt(
                POLY_ALPHA(31) | POLY_CULL_NONE | POLY_ID(0));
        }
        drawCube(2.9f, 0.0f, 0.0f, 1.0f, 0.0f, 0.0f);
        if(clicked==GREEN) {
            glPolyFmt(
                POLY_ALPHA(31) | POLY_CULL_NONE | POLY_ID(1));
        } else {
            glPolyFmt(
                POLY_ALPHA(31) | POLY_CULL_NONE | POLY_ID(0));
        }
        drawCube(-3.0f, 1.8f, 2.0f, 0.0f, 1.0f, 0.0f);
        if(clicked==BLUE) {
            glPolyFmt(
                POLY_ALPHA(31) | POLY_CULL_NONE | POLY_ID(1));
        } else {
            glPolyFmt(
                POLY_ALPHA(31) | POLY_CULL_NONE | POLY_ID(0));
        }
        drawCube(0.5f, -2.6f, -4.0f, 0.0f, 0.0f, 1.0f);
    }
    glPopMatrix(1); // Restaura modelview a donde fue rotada
    // Dibuja la escena recortada
    {
        clicked = NOTHING; // Reset de clicked
        closeW = 0x7FFFFFFF; // Reset de la distancia
        // Deja la vista fuera de la pantalla, así se oculta
        // todos los render que se hagan durante esta fase
        glViewPort(0,192,0,192);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        // Renderiza sólo lo que esté bajo el cursor
        gluPickMatrix(touchX,(191-touchY),4,4,viewport);
        // La perspectiva debe ser la misma que la original
        gluPerspective(30, 256.0 / 192.0, 0.1, 20);
```

```
                glMatrixMode(GL_MODELVIEW);
                startCheck();
                drawCube(2.9f, 0.0f, 0.0f, 1.0f, 0.0f, 0.0f);
                endCheck(RED);
                startCheck();
                drawCube(-3.0f, 1.8f, 2.0f, 0.0f, 1.0f, 0.0f);
                endCheck(GREEN);
                startCheck();
                drawCube(0.5f, -2.6f, -4.0f, 0.0f, 0.0f, 1.0f);
                endCheck(BLUE);
            }
        }
        glPopMatrix(1); // Restaura modelview
        return TRUE;
}
void drawCube(float x, float y, float z, float r, float g, float b){
        glTranslate3f32(floattof32(x),floattof32(y),floattof32(z));
        glColor3f(r,g,b);
        glBegin(GL_QUADS);
        // Cara frontal
        glVertex3f(-1.0f, -1.0f, 1.0f);
        glVertex3f( 1.0f, -1.0f, 1.0f);
        glVertex3f( 1.0f, 1.0f, 1.0f);
        glVertex3f(-1.0f, 1.0f, 1.0f);
        // Cara trasera
        glVertex3f(-1.0f, -1.0f, -1.0f);
        glVertex3f(-1.0f, 1.0f, -1.0f);
        glVertex3f( 1.0f, 1.0f, -1.0f);
        glVertex3f( 1.0f, -1.0f, -1.0f);
        // Cara superior
        glVertex3f(-1.0f, 1.0f, -1.0f);
        glVertex3f(-1.0f, 1.0f, 1.0f);
        glVertex3f( 1.0f, 1.0f, 1.0f);
        glVertex3f( 1.0f, 1.0f, -1.0f);
        // Cara inferior
        glVertex3f(-1.0f, -1.0f, -1.0f);
        glVertex3f( 1.0f, -1.0f, -1.0f);
```

```
        glVertex3f( 1.0f, -1.0f, 1.0f);
        glVertex3f(-1.0f, -1.0f, 1.0f);
        // Cara derecha
        glVertex3f( 1.0f, -1.0f, -1.0f);
        glVertex3f( 1.0f, 1.0f, -1.0f);
        glVertex3f( 1.0f, 1.0f, 1.0f);
        glVertex3f( 1.0f, -1.0f, 1.0f);
        // Cara izquierda
        glVertex3f(-1.0f, -1.0f, -1.0f);
        glVertex3f(-1.0f, -1.0f, 1.0f);
        glVertex3f(-1.0f, 1.0f, 1.0f);
        glVertex3f(-1.0f, 1.0f, -1.0f);
        glEnd();
}
```

Apéndice C

Manual de usuario

En este manual se explicara al usuario que juegue a QUIZ LIBRE[9], como debe manejar, configurar e instalar, dicho juego desde sus controles hasta la explicación de las diferentes secciones que compone el menú, así como la explicación de añadir nuevas preguntas al juego.

C.1. Instalación del juego.

En primer lugar este juego no se puede ejecutar desde un emulador, por ello se debe disponer de un cartucho que pueda ejecutar aplicaciones caseras en la consola con una memoria interna para el almacenamiento de los juegos.

El juego está compuesto de dos archivos:

- **QUIZ LIBRE.NDS**: Fichero que contiene el juego al completo con formato que puede ejecutar la aplicación de la Nintendo DS.
- **Carpeta data y fichero DB.db**: El fichero DB.db contiene la base de datos del juego en formato sqlite[11].

Tanto la carpeta como el fichero del juego se deben poner en la raíz de la tarjeta de memoria del cartucho, para que el software de la tarjeta pueda encontrar el juego correctamente y ejecutarlo. Hay que tener muy en cuenta que la carpeta junto a la base de datos deben estar bien colocados en la raíz, ya que si no está en el sitio correspondiente, el juego no puede cargar la base de datos.

C.2. Configuración del juego

El único apartado del juego configurable actualmente es el apartado de las preguntas que componen los temarios. Hay que tener en cuenta que durante el juego se realiza una selección de temario y es

lo que determina las categorías que van a componen la partida, y las preguntas a realizar por las categorías.

Para la configuración de las categorías de los temarios, se debe introducir en la carpeta "Database" del programa de gestión de la base de datos para que el programa reconozca la los datos y puedan realizarse los cambios.

Una vez cargado el programa con la base de datos incluida, se puede agregar información a la misma.

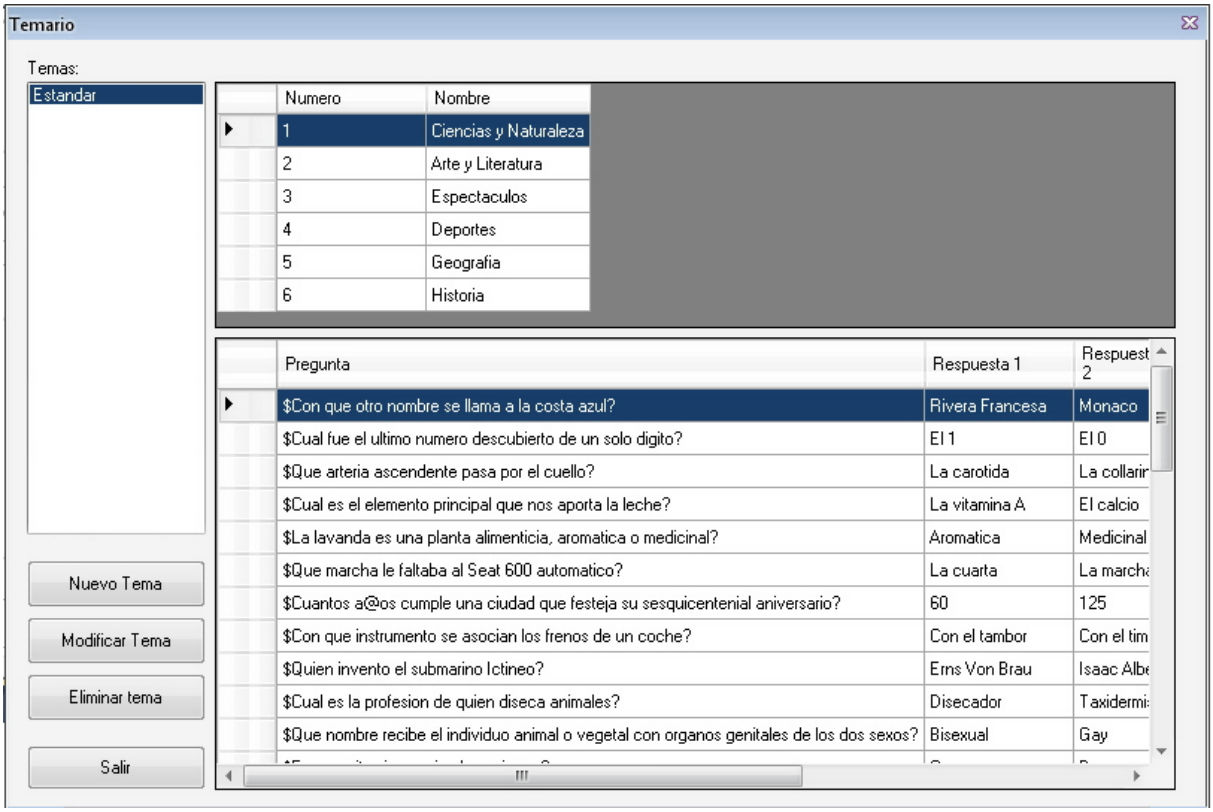


Figura C.1: Captura del programa de gestión. Categorías del temario

En el formulario inicial se pueden realizar tres acciones:

- **Nuevo Tema**: Agregar un nuevo tema a la base de datos vacío.
- **Modificar Tema**: Modificar un tema existente.
- **Eliminar Tema**: Elimina un temario con todas las categorías y preguntas existentes. Hay que tener cuidado de no eliminar accidentalmente ya que no se podrá recuperar.

Tanto "modificar tema" como "añadir tema", el programa nos lleva a un nuevo formulario de gestión de las categorías de los temarios. En este nuevo formulario se pueden realizar las siguientes acciones:

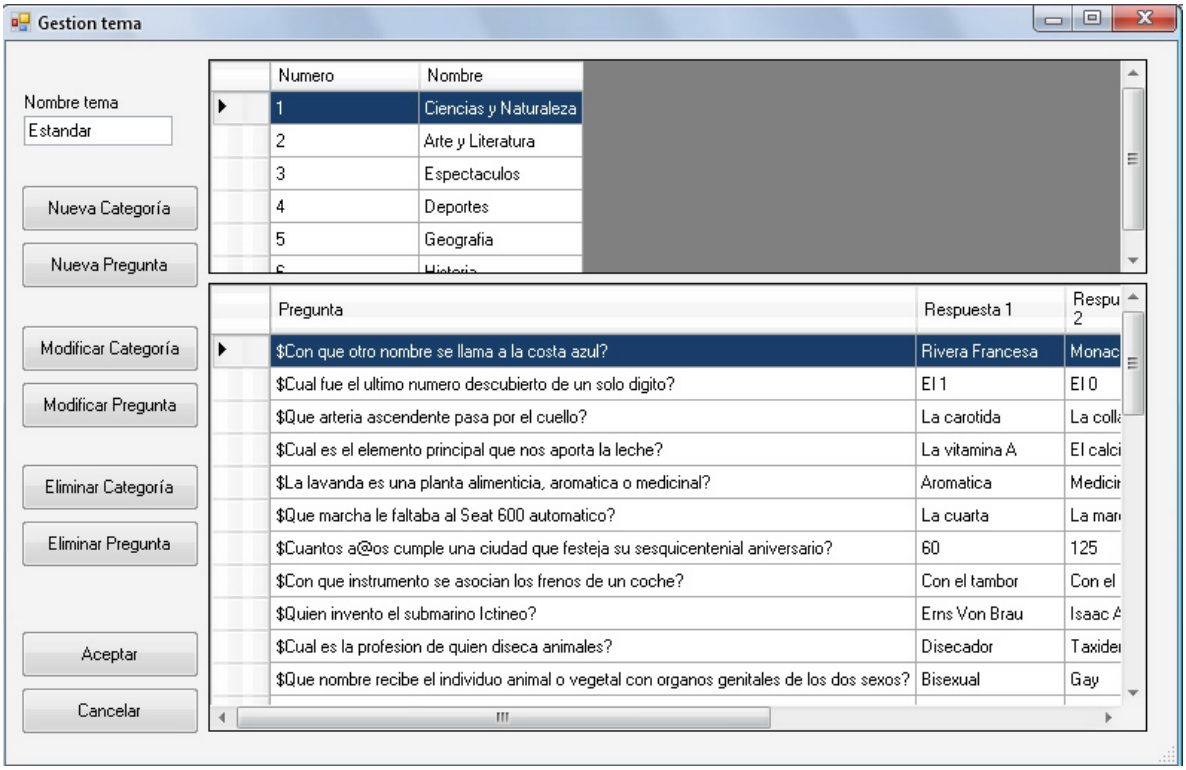


Figura C.2: Formulario de gestión de preguntas y categorías

- **Nueva Categoría**: Agrega una nueva categoría al temario.
- **Modificar Categoría**: Modificar el nombre de la categoría seleccionada.
- **Eliminar Categoría**: Elimina una categoría con todas las preguntas asignadas de la base de datos.
- **Nueva Pregunta**: Abre un nuevo formulario para añadir la información de los textos de la pregunta.
- **Modificar Pregunta**: Muestra un formulario con los textos de la pregunta seleccionada para que se modifique.
- **Eliminar Pregunta**: Elimina la pregunta seleccionada de la base de datos.

Hay que tener en cuenta que un temario no puede tener menos de seis categorías pero puede tener tantas categorías que se quiera, eso si, el tablero solo cargara las seis primeras categorías.

Una vez se le da a aceptar se actualiza la información del temario con la información de las categorías.

C.3. Manual del juego

Ahora se procederá la explicación que un usuario puede realizar durante el transcurso del juego.

C.3.1. Durante el menú principal del juego.

Controles para manejar el menú

- **Pad Control**: Moverse por las diferentes secciones del menú.
- **Botón A**: Seleccionar una opción y avanzar a la siguiente pantalla del menú.
- **Botón B**: Cancelar una acción y volver a la pantalla anterior.
- **Pulsación stylus**: Selecciona acciones si se clica en los diversos botones que contiene la pantalla.

Las opciones del menú principal

Figura C.3: Menú Principal del juego

- **Nueva Partida**: Se realiza el proceso de creación de una nueva partida por el usuario, seleccionando tablero, temario y cantidad y nombres de jugadores de la partida.
- **Cargar Partida**: Se carga desde la base de datos una partida guardada con anterioridad. Solo se puede guardar una partida a la vez así que directamente se pasa a la partida. Si la base de datos no tiene ninguna partida se vuelve al menú principal.
- **Ranking**: Muestra los diez jugadores con mayor puntuación de juegos anteriores. En esta pantalla, se regresa al menú principal pulsando Botón A o pulsando con le stylus en la pantalla.
- **Créditos**: Muestra un pequeño texto de referencia del juego y agradecimientos del creador. Se regresa al menú principal pulsando botón A o pulsando con le stylus en la pantalla.

C.3.2. Durante la partida

Controles del jugador

- **Pulsacion stylus**: se utiliza tanto para seleccionar la casilla a la que se desea mover la ficha como para la selección de la respuesta correcta.
- **Pad de control:** Se usa para moverse por las diferentes opciones de los menús y mover la pantalla por el tablero.
- **Botón A**: Seleccionar una opción.
- **Botón "Start"**: Muestra el menú de pausa del juego.
- **Botón "Select"**: Muestra una pantalla que indica el nombre de las categorías según el color de la casilla.

Casillas del tablero

Figura C.4: Tablero del juego

Los colores de las casillas indican la categoría que pertenece, así que si la ficha cae en una casilla se procederá a la formulación de la pregunta. Para saber que casillas están disponibles para moverte, están marcadas con una luz encima.

Casillas especiales:

- **Casilla blanca**: Si una ficha selecciona esta casilla simplemente no se formula ninguna pregunta y se vuelve a tirar los dados.
- **Casilla central**: Esta casilla hace lo mismo que las casillas blancas.
- **Casillas Principales**: Casillas puestas en los extremos de los caminos centrales, mas grandes que lo habitual, que indican que si el jugador responde correctamente la pregunta, conseguirá una porción del queso con el color de la casilla, si no lo ha conseguido antes.

Menú Pause

Figura C.5: Menú de pause del juego.

Este menú se accede pulsando el botón *Start* de la consola. En este menú hay tres opciones que se pueden realizar:

- Continuar Partida: Se sale del menú continuando el transcurso de la partida.
- Guardar Partida: Se guarda en la base de datos la información de la partida y se sale al menú principal.
- Salir: La partida se finaliza y se vuelve al menú principal.

Pantalla de la pregunta

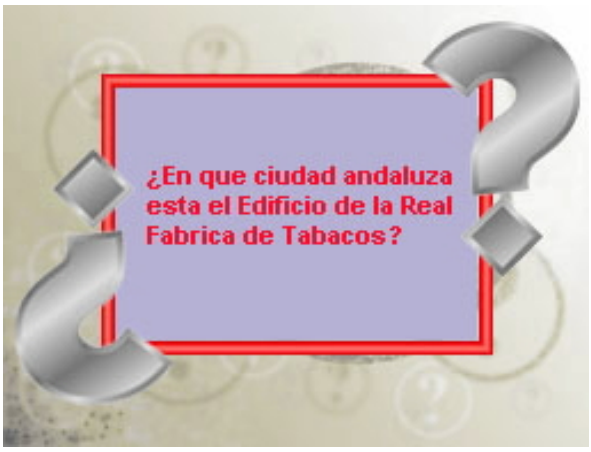


Figura C.6: Pantalla de la pregunta del juego.

Figura C.7: Pantalla de respuestas de una pregunta.

En esta pantalla hay ciertos elementos a tener en cuenta. En la pantalla superior tenemos el texto de la pregunta seleccionada que debemos responder, y en la pantalla inferior tenemos las tres respuestas posibles. Se deben pulsar con el stylus la respuesta que creemos que será la correcta y el juego nos dice, si hemos acertado o no.

En la parte superior derecha de la pantalla inferior se muestra un círculo de colores que corresponde al reloj del juego. Este círculo ira eliminando colores cada 5 segundos hasta un total de 60 segundos, si el jugador no ha respondido en ese tiempo se da la pregunta como incorrecta.

Puntuaciones

Figura C.8: Pantalla de puntuaciones

El sistema de puntuaciones está configurado de la siguiente manera.

- Respuesta correcta:
  - Casilla normal: Se suma diez puntos a la puntuación del jugador
  - Casilla principal: Se suma treinta puntos a la puntuación del jugador
- Respuesta incorrecta: Se resta diez puntos a la puntuación del jugador.

Finalización de la partida

La partida se da por concluida cuando cualquier jugador ha conseguido la última porción de queso que le quedaba. Entonces el jugador que tiene el turno es el ganador de la partida.

Si el jugador ha conseguido una puntuación mejor que alguno de los diez primeros del ranking se agrega a la lista.

Apéndice D

GNU Free Documentation License

Version 1.3, 3 November 2008

D.1. Preamble

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

D.2. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "**Modified Version**" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "**Secondary Section**" is a named appendix or a front-matter section of theDocument that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "**Invariant Sections**" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "**Cover Texts**" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "**Transparent**" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "**Title Page**" means, for a printed book, the title page itself, plus such following pages as

are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "**publisher**" means any person or entity that distributes copies of the Document to the public.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may includeWarranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

D.3. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

D.4. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with

each Opaque copy a computer-network location from which the general network-using public has access to download using publicstandard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

D.5. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

1. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
2. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
3. State on the Title page the name of the publisher of the Modified Version, as the publisher.
4. Preserve all the copyright notices of the Document.
5. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
6. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
7. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
8. Include an unaltered copy of this License.
9. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

10. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

11. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

12. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

13. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

14. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

15. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

D.6. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

D.7. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

D.8. AGGREGATION WITH INDEPENDENTWORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

D.9. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in

addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

D.10. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

D.11. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

D.12. RELICENSING

"Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

"CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

"Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

D.13. ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright c YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with . . . Texts." line with this:

> with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.